Agents Mobiles : itinéraires pour l'administration système et réseau

Emmanuel Reuter

Université de Nice Sophia Antipolis
UFR SCIENCES

École Doctorale : STIC

THÈSE

Présentée pour obtenir le titre de :
Docteur en SCIENCES
de l'Université de Nice Sophia Antipolis

Spécialité : Informatique

par

Emmanuel Reuter

Équipe d'accueil : OASIS - INRIA Sophia Antipolis

Titre de la thèse :

Agents Mobiles : Itinéraires pour l'administration système et réseau

Thèse dirigée par Françoise BAUDE
Soutenue le 28 Mai 2004

Président			
M. :	Michel	Riveill	Univ. Nice Sophia Antipolis
Rapporteurs			
MM. :	Olivier	Festor	INRIA Lorraine
	Serge	Chaumette	Univ. Bordeaux 1
Examinateurs			
MM. :	Françoise	Baude	Univ. Nice Sophia Antipolis
	Jean-Luc	Ernandez	Ingénieur ATOS Origin

Je remercie avant tout Françoise Baude qui m'a encadré durant ces quatre dernières années et sans qui cette thèse n'aurait pas pu se faire. Elle a su me guider par ses nombreux conseils, les encouragements mentionnés à chaque fin d'emails et pendant les nombreux repas de travail que nous nous sommes fixés avec une certaine régularité.

L'accueil qui m'a été réservé dans l'équipe OASIS a été à la fois chaleureux et enrichissant scientifiquement et m'a permis de m'ouvrir à de nouveaux sujets. J'ai beaucoup apprécié les échanges entre Julien Vassière, Fabrice Huet, Romain Quilici, membres de l'équipe Oasis, qui m'ont permis d'avancer dans mon travail de recherche tout en occupant un poste à l'IUFM de l'académie de Nice.

Je tiens à remercier bien entendu les membres du jury : Michel Riveill qui a accepté de présider ce jury, Olivier Festor, Serge Chaumettre pour avoir accepté d'être rapporteurs et enfin Jean-Luc Ernandez pour avoir accepté d'être un examinateur de cette thèse.

Je tiens à remercier les deux directeurs de l'IUFM, François Rocca et René Lozi, qui m'ont autorisé successivement à continuer ma thèse en parallèle du travail d'ingénieur d'études que j'exerce au sein de cet établissement. Plus particulièrement François Rocca qui a été pour moi l'étincelle de départ de cette grande aventure.

Une pensée particulière est adressée à Richard MANAS, Ingénieur de Recherche au Centre Informatique de l'Université de Nice Sophia Antipolis, pour son aide et les discussions que nous avons eues pendant ces quatres années.

Je n'oublie pas le personnel de l'informatique de l'IUFM qui a su m'aider dans mes démarches de tests et qui a parfois supporté mes absences pendant que je voguais vers d'autres cieux, ceux de la recherche bien sûr.

A mon pays, la Nouvelle-Calédonie.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Contexte

De nos jours, toutes les entreprises sont équipées au minimum d'un réseau local, et pour les plus importantes d'entre elles de réseaux longue distance (Wide Area Network). L'administration de réseaux est devenue un point critique dans toutes ces entreprises par l'hétérogénéité même des éléments composant le réseau. Du fait de cette hétérogénéité, il devient indispensable de surveiller en permanence les éléments-clefs du réseau, afin que les utilisateurs ne soient pas affectés par des incidents de fonctionnement et que la perte d'exploitation soit la plus faible possible en cas d'incident. Pour cela, les entreprises investissent dans des outils d'administration de réseau très onéreux [35, 37] et qui ne sont pas nécessairement adaptés à chacune d'entre elles.

Un système d'administration de réseau, qui fournit des mécanismes de surveillance, est composé d'une ou plusieurs stations de supervision qui communiquent entre elles et avec les éléments du réseau par un protocole d'administration de réseau, les plus connus étant SNMP [87] et CMIP [104]. Il est nécessaire d'arriver à le répartir dans les sous-réseaux de l'entreprise pour diminuer l'utilisation de la bande passante sur les liens inter-réseaux, d'améliorer la réactivité du système d'administration en le distribuant sur chaque sous-réseaux de l'entreprise.

1.2 Motivation

Il y a de plus en plus d'éléments dans les réseaux des entreprises qu'un administrateur doit surveiller. Pour cela il dispose, normalement, d'outils pour analyser son réseau, surveiller son système, mais une plate-forme à agents mobiles peut apporter une aide substantielle à un administrateur. En effet, certaines tâches routinières, comme par exemple l'effacement de fichiers temporaires ou le déploiement de logiciels, peuvent être directement effectuées par des agents mobiles

et relativement autonomes. La programmation de tels agents mobiles peut être simplifiée dès lors qu'un cadre d'administration de systèmes et de réseaux fournissant les outils appropriés est disponible. Par exemple, l'inventaire automatisé du parc informatique est une opération facilement réalisable avec une technique à agents mobiles par laquelle on déploit un ou plusieurs agents autonomes. Le mécanisme de migration fourni dans les plates-formes à agents mobiles permet de faire déplacer sans souci les agents mobiles et de faire exécuter automatiquement une tâche pré-programmée adaptée à une opération d'administration localement sur chaque élément. Il est à présent admis que les agents mobiles dans un cadre d'administration système et réseau permettent la réduction du temps de traitement des messages échangés entre des éléments du réseau, permettent de limiter la bande passante nécessaire aux opérations d'administration distantes, et enfin le traitement local des informations permet de réagir plus rapidement aux changements dans le réseau [64]. Par ailleurs, la technologie des agents mobiles semble aussi adaptée à l'administration des réseaux en émergence que sont les *active networks*, notamment en facilitant le déploiement des tâches d'administrations dans ce nouveau type de réseau [75].

Pour utiliser des agents mobiles dans un cadre d'administration système et réseau, la mise en œuvre d'itinéraires permettant l'exécution de tâches d'administration est un besoin pour rendre complètement autonomes ces agents mobiles pendant l'exécution des tâches pré-programmées. Idéalement, un itinéraire dynamiquement créé par la connaissance automatique de l'infrastructure du réseau, serait adapté pour suivre l'évolution des réseaux et des services que ces réseaux peuvent fournir et pour déployer dynamiquement, en parallèle si nécessaire, les agents mobiles d'administration ayant en charge la supervision de tels réseaux. Un tel concept d'itinéraire dynamique et intelligent pour son suivi, offrirait plus d'avantages qu'un itinéraire statique, qui plus est défini manuellement : l'itinéraire serait toujours à jour, et son suivi serait adapté aux conditions réseaux environnantes, c'est-à-dire prenant en compte des pannes dans le réseau, les nouveaux éléments, et ce de manière complètement transparente à l'administrateur.

Donnons un exemple d'itinéraire que l'on peut appliquer au genre humain. Imaginons un voyageur partant de Nice à destination de Nouméa, pour aller s'installer en Nouvelle-Calédonie. Décrivons plus précisément son itinéraire de voyage. A l'aéroport de Nice-Côte d'Azur, notre voyageur se rend au comptoir de la compagnie afin de récupérer son titre de transport. Il prend son premier avion à destination de Paris, départs Internationaux. A l'aéroport de Roissy, il récupère des présents pour ses amis néo-calédoniens et les emportent avec lui dans l'avion qui l'amène à Osaka, aéroport de transit. Profitant de la baisse du cours du Yen, notre voyageur acquiert un appareil photo de bonne qualité afin de ramener des souvenirs de son voyage. Arrivé à destination, notre voyageur offre les présents à ses amis sur le territoire. Néanmoins, notre voyageur n'oubliera pas de rester en contact avec sa ville de départ.

Revenons à notre agent mobile. En se basant sur le descriptif de l'itinéraire de

voyage, notre agent mobile se rend de sa station de départ (Nice) vers la station Roissy, premier point d'arrêt. Sur la station Roissy, il exécute une première collecte de données avant de reprendre son chemin vers la station Osaka. Deuxième point d'arrêt, pour lequel l'agent mobile prend une image de l'état du système. L'agent mobile se dirige ensuite vers sa station d'arrivée, Nouméa, où il termine son itinéraire. Même à distance, lui et sa station d'origine peuvent rester en communication et ainsi notre agent mobile peut transmettre les informations qu'il a collectées pendant son itinéraire.

Pour généraliser une telle solution, un administrateur doit être en mesure de fournir un itinéraire d'administration aux agents mobiles. La création de ces itinéraires peut être statique, c'est-à-dire des itinéraires définis manuellement réseau par réseau. Une solution généralisable qui donnerait des itinéraires dynamiques et adaptables selon les réseaux et les éléments qui y sont connectés est plus appropriée, à cause des changements incessants qui surviennent dans les réseaux. Pour ce faire, la connaissance de l'infrastructure interconnectant les éléments entre eux est nécessaire, typiquement la topologie des différents réseaux constituant le réseau de l'entreprise.

1.3 Contribution et organisation du manuscrit

Dans un premier temps nous détaillons le métier de l'administration des systèmes et de réseaux afin de mieux cerner l'intérêt et les difficultés d'un tel métier au sein des entreprises (chapitre 2).

L'émergence des plates-formes à agents mobiles et leur intégration possible dans le monde de l'administration système et réseau permettent d'envisager des solutions pour aider un administrateur dans le métier qu'il exerce jour après jour, afin de satisfaire aux contraintes d'exploitation des systèmes et des réseaux dont il a la charge. Une analyse comparative détaillée de telles plates-formes est présentée dans le chapitre 3. Elle permet de mettre en évidence l'intérêt que pourrait avoir le fait de disposer d'itinéraires d'administration qui puissent être construits dynamiquement plutôt que statiquement.

Ce travail de thèse consiste donc à proposer la définition d'un mécanisme de fabrication puis d'utilisation d'itinéraires dynamiques pour l'administration système et réseau. Ces itinéraires, définis dynamiquement, et dans un premier temps sous-réseau par sous-réseau, sont des itinéraires locaux qui sont par la suite combinés dynamiquement entre eux de sorte à pouvoir prendre en considération l'intégralité de l'architecture à superviser. Nous validons cette définition en fournissant une plate-forme complète ainsi qu'un cadre de programmation d'agents mobiles d'administration système et réseau. Pour ce faire, nous nous basons sur la bibliothèque ProActive pour le calcul parallèle, réparti et mobile.

En nous basant sur le modèle de programmation d'agents mobiles qu'offre la

bibliothèque ProActive, nous décrivons quelles extensions nous y avons apportées dans le but de définir un mécanisme d'itinéraires pour l'administration système et réseau (chapitre 4).

La conception d'une plate-forme d'administration à base d'agents mobiles ProActive mettant en œuvre des itinéraires d'administration est décrite en termes de services qu'il est nécessaire de mettre en place (chapitre 5). Une présentation détaillée d'un algorithme de découverte de la topologie est ensuite donnée. Cet algorithme a pour objectif de fournir à la plate-forme d'administration développée des listes d'éléments découverts, afin de permettre la programmation aisée d'itinéraires d'administration pour des agents mobiles d'administration système et réseau.

Une fois une telle plate-forme disponible, nous sommes en mesure d'expliciter en détails comment programmer aisément des agents mobiles d'administration système et réseau (chapitre 6).

Le chapitre 7 a comme intérêt de valider cette plate-forme et son applicabilité. On fournit des exemples réels d'implantation d'agents d'administration et des mesures et des analyses de performances comparatives avec de l'administration classique SNMP centralisée, et ce pour une large gamme de configuration de réseau.

Nous concluons par un récapitulatif de la contribution que nous avons apportée quant à la mise en œuvre des itinéraires d'administration, et donnons les perspectives associées à une telle mise en œuvre : supervision d'une plate-forme à agents mobiles, supervision et aide à la répartition des agents mobiles sur une grille de calculs, en sont des perspectives possibles.

Chapitre 2

L'administration des systèmes et des réseaux

2.1 Motivation

L'évolution des moyens de communication mis à la disposition des entreprises, l'hétérogénéité de l'infrastructure des réseaux et des éléments qui les composent, la quantité d'informations dont veulent disposer les entreprises, tout cet ensemble nécessite d'apporter une cohérence dans le fonctionnement de l'infrastructure de communication.

Pour ce faire, et malgré la diversité des éléments constituant le cœur du réseau (par exemple les équipements actifs) proposés par les constructeurs, l'administration des systèmes et des réseaux repose fondamentalement sur la connaissance de l'infrastructure des réseaux et sur celle du fonctionnement des systèmes informatiques entrant en jeu. Le fonctionnement de ces systèmes doit être assuré par un ou plusieurs administrateurs, en fonction du nombre d'éléments qui interviennent au sein de l'infrastructure de communication. Ces administrateurs peuvent travailler de concert pour assurer la maintenance des différents matériels, la mise-à-jour des logiciels, la préparation et l'évolution des infrastructures et la mise en service des nouveaux équipements. La connaissance du fonctionnement des réseaux permet aux administrateurs de choisir le type de techniques adaptées afin d'en assurer la pérennité (par exemple doublage des équipements actifs, liens de secours, liens hertziens, etc..).

Pour obtenir un fonctionnement quasi-permanent des systèmes et des réseaux, les administrateurs doivent pouvoir récupérer à tout instant un rapport sur l'état du fonctionnement du réseau et de celui des systèmes qui y sont connectés. Cela implique une collecte des informations disponibles dans le réseau pour déterminer son état. Les administrateurs peuvent déterminer, par la collecte des informations, les défaillances possibles et les surcharges pouvant amener à un dysfonctionnement. Comme la quantité d'informations recueillie est importante, il va de soi que

cette tâche doit être automatisée pour que l'administrateur n'ait que de l'information utile pour effectuer son travail. C'est dans cette optique que les stations et les logiciels d'administration ont été créés. Ces logiciels donnent une vue complète du réseau et ils permettent à l'administrateur (ou aux administrateurs) une intervention rapide et efficace.

Dans la section 2.2 nous décrirons les tâches des administrateurs et les couches du modèle OSI entrant en jeu dans un système d'administration de réseau. Le kit de survie de l'administrateur réseau, c'est-à-dire les outils de base utiles à un administrateur, sera présenté dans la section 2.3

Par la suite nous définirons dans la section 2.4 les protocoles d'administration de réseau et les outils les plus usités pour réaliser des tâches d'administration. Toutefois, les outils les plus usités dans le métier de l'administration réseau fournissent des informations qui ne peuvent malheureusement pas être prises en compte automatiquement par d'autres outils ou plates-formes que l'administrateur souhaiterait mettre en œuvre.

2.2 Définition de l'administration des systèmes et des réseaux

L'administration des systèmes et des réseaux consiste à contrôler, coordonner et surveiller les différentes ressources mises en œuvre afin de fournir des services opérationnels aux utilisateurs : ces ressources sont les équipements, le réseau, les services sont ceux offerts par les différents serveurs, les applications, Internet, etc... Par exemple, ces services peuvent être :

- l'accès aux données et aux ressources informatiques partagées (base de données, annuaires, etc)
- la consultation des sites Internet ; l'accès au réseau informatique

Toute entreprise possédant un grand nombre d'ordinateurs, emploie des administrateurs de réseau et de systèmes pour assurer la qualité et la continuité du service.

Le modèle de l'administration de réseau selon la classification SMFA (Specific Management Functional Areas [29]) consiste en 5 domaines de compétence : la gestion des performances, la gestion des fautes, la gestion de la configuration, la gestion des informations comptables, la gestion de la sécurité.

2.2.1 Gestion des performances

Un administrateur réseau a comme responsabilité la mise en place, le suivi et l'évolution des moyens de communication. Compte tenu de l'augmentation régulière du trafic sur un réseau local, par exemple, un administrateur doit envisager le changement de l'infrastructure réseau afin d'améliorer le débit offert aux utilisateurs. En fonction de la charge du réseau qu'un administrateur supervise,

il peut prévoir le changement de la structure de communication du réseau (cf. figure 2.1) en introduisant une segmentation du trafic par l'ajout d'un nouvel équipement actif (par exemple, un commutateur). La conséquence directe sera l'amélioration du trafic dans le cœur du réseau et une meilleure segmentation du trafic (cf. figure 2.2) dans chaque sous-réseau connecté.

Pour superviser les moyens de communication, l'administrateur peut disposer d'outils d'analyse (par exemple Multi Router Traffic Grapher [56], HP Network Node Manager [35]) afin de suivre en temps réel les différents flux qui circulent sur les réseaux dont il a la charge. De tels outils d'analyse permettent d'effectuer des prospectives d'évolution de l'infrastructure physique tant sur le point de vue du changement de débit que sur le type de liaison utilisée (par exemple : une liaison spécialisée plutôt qu'une liaison point-à-point Numéris active à la demande).

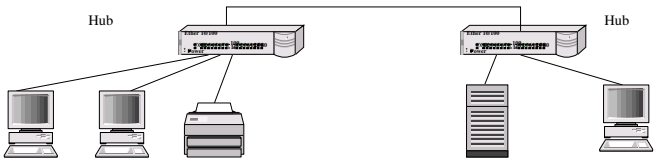

FIG. 2.1 – Réseau Ethernet non segmenté

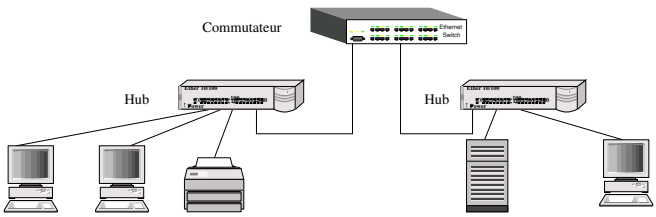

FIG. 2.2 – Réseau Ethernet segmenté

2.2.2 Gestion des fautes

Quels que soient les systèmes informatique dont l'administrateur a la charge, chacun de ces systèmes ne peut être exempt d'erreurs. Le travail de l'administrateur consiste à assurer le fonctionnement de ces systèmes. Par exemple, la panne d'un commutateur dans le réseau est un événement rare mais très important dont l'administrateur doit tenir compte dans ses procédures de gestion des incidents. L'administrateur doit être en mesure de pouvoir prendre les décisions nécessaires afin de corriger dans un temps très bref la défaillance de son système. Les procédures mises en place pour assurer la continuité de service ou un fonctionnement en mode dégradé sont des éléments nécessaires pour que les usagers de l'infrastructure de communication soient le moins pénalisés par des pannes sur le réseau.

2.2.3 Gestion de la configuration

Un administrateur de systèmes informatique a pour tâche l'installation, la mise en route de services réseaux et la maintenance de ces différents systèmes. L'administrateur assure aussi la configuration, la mise à niveau des équipements actifs de l'infrastructure du réseau et il se trouve en mesure de prévoir les différentes opérations de maintenance nécessaires. Par exemple, le changement de la version d'un système d'exploitation sur un commutateur faisant suite à un avis du constructeur de l'équipement actif est une tâche qui lui incombe et qu'il doit mettre en œuvre.

L'administrateur est responsable de l'adressage des machines sur le réseau, il suit l'évolution du parc de machines connectées et il organise, si nécessaire, le redéploiement des systèmes sur son infrastructure de communication.

Pour être aidé dans sa tâche de gestion, l'administrateur peut disposer en fonction du type du système d'exploitation de ses systèmes, d'outils divers et variés mais dont l'usage reste souvent propriétaire à un type de plate-forme (WinNT, Solaris, etc..). Par exemple, Webmin [16] fonctionnant sous Linux permet à un administrateur d'effectuer de la maintenance ou de la configuration de son système.

2.2.4 Gestion des informations comptables

Un administrateur peut fournir aux instances dirigeantes de son entreprise les coûts engendrés par telle ou telle application. Pour cela, l'administrateur réalise des calculs de coûts en fonction des débits réseaux utilisés, de la charge des machines sur lesquelles tournent les applications. L'administrateur doit disposer pour cela d'outils pour mesurer le trafic, calculer le taux de charge des applications. Par exemple, les applications Internet payantes (`http://link.springer.de`) sont à la charge de l'administrateur qui doit assurer pour son entreprise la comptabilisation des accès aux services fournis et permettre l'ouverture de nouveaux accès à ces mêmes services.

2.2.5 Gestion de la sécurité

Un administrateur système et un administrateur réseau doivent travailler de concert pour la mise en œuvre de nouveaux services. Les administrateurs doivent assurer la sécurité des informations et la sécurité des accès. Par exemple, un annuaire d'entreprise doit pouvoir être consulté mais il convient de s'assurer que la population ayant accès à ces informations soit une population limitée : un Internaute quelconque ne bénéficierait pas des mêmes droits d'accès qu'un employé de l'entreprise hébergeant ce service. Il s'agit donc d'avoir des procédures de mise en route de nouveaux services tout en préservant la sécurité et la fonctionnalité des services déjà existant.

2.3 Le kit de survie de l'administrateur réseau

L'objectif de cette section est de présenter les couches du modèle OSI qui sont directement liées au métier de l'administrateur réseau : la couche physique, la couche liaison de données et la couche réseau, toutes les trois intervenant dans le fonctionnement d'un réseau. Nous donnerons les détails des concepts et des outils qu'il est nécessaire de connaître afin de pouvoir comprendre le métier de l'administrateur de réseau.

Définition 1 *Équipements actifs : Le terme équipements actifs, sous entend dans ce manuscrit, les éléments qui sont connectés sur un réseau, qui participent à son fonctionnement et qui sont administrables, c'est-à-dire équipés d'un agent SNMP.*

Définition 2 *Réseau : Le réseau est l'ensemble de voies de communication, conducteurs électriques, etc., qui desservent une même unité géographique, dépendant de la même entreprise. Nous nous baserons sur cette définition pour nous rapprocher plus précisement du* réseau *que nous utilisons tous les jours, c'est-à-dire un moyen de communication et d'échange entre différentes entités.*

Ainsi, le réseau en tant qu'entité communiquante est la partie principale de notre préoccupation puisque celle-ci se situe au cœur de tout le trafic qui y circule. Il va de soi qu'il faut donc une architecture physique afin de relier les entités le constituant et des protocoles de communication pour permettre à ces différentes entités de dialoguer entre elles. Pour définir les différents niveaux permettant à ces entités de dialoguer entre elles, l'ISO (organisation internationale de normalisation) a donc défini un modèle en sept couches appelé modèle de référence (modèle OSI : Open Systems Interconnection). Ce modèle en sept couches est défini comme suit : Physique, Liaison, Réseau, Transport, Session, Présentation, Application.

Dans ce modèle et en rapport avec l'administration des systèmes et des réseaux, nous nous préoccuperons principalement de deux couches du modèle OSI : la couche liaison et la couche réseau [91]. Effectivement, ces deux couches ont une très grande importance puisqu'elles relèvent activement du fonctionnement du réseau, et qu'il est nécessaire d'assurer leur disponibilité et leur fiabilité.

2.3.1 La couche physique

C'est la couche qui s'occupe de la transmission des bits d'information de façon à ce que le récepteur des données recueille une information non altérée par le canal de communication.

La partie physique d'une architecture réseau dépend de la qualité de service que l'on souhaite obtenir. Plusieurs types de couche physique existent, comme la partie en câble coaxial (10Base-2 ou 10Base-5 [1]), le câble téléphonique à

paires torsadées (10BaseT ou 100BaseT [1]), les connexions en fibre optique, les connexions par voie hertzienne, etc.., tout ce qui permet de faire circuler de l'information. La qualité de l'installation de cette couche est primordiale puisqu'elle a une incidence sur le fonctionnement même du réseau. Par exemple, un connecteur RJ45 mal serti engendrerait des perturbations sur le réseau (collision, tentative pour trouver la vitesse du média, etc..).

Le contrôle de la qualité de cette liaison peut se faire par exemple dans le cas d'un réseau 10Base-T, par l'utilisation d'un testeur de câbles. Cette technique permet de se rendre compte des problèmes de connectivité électrique sur le média utilisé.

2.3.2 La couche liaison de données

La tâche principale de cette couche est de prendre un moyen de communication brut et de le transformer en une liaison qui paraît exempte d'erreurs de transmission à la couche réseau (on parle de trames). Quelques types de protocoles sont utilisés pour assurer le fonctionnement du canal de communication comme le protocole CSMA/CD [60], l'anneau à jeton [61], ou de façon plus évoluée l'ATM (Asynchronous Transfert Mode [5]).

Pour vérifier la qualité du canal de communication offerte par la couche de liaison dans le cas d'une topologie réseau en étoile, une technique simpliste mais assez efficace consiste à utiliser deux ordinateurs connectés sur un concentrateur (Hub) ou un commutateur (Switch) et d'appliquer un outil client/serveur qui émet des trames depuis un poste à destination de l'autre. Si dans ce cas les trames émises sont reçues par l'autre ordinateur, alors le canal de communication peut être considéré comme fiable.

Nous allons détailler les réseaux de type *Ethernet* [1]. Ces réseaux sont les plus courants et la technologie *Ethernet* a permis une large diffusion de ces réseaux dans les entreprises. Dans ce type de réseau, les stations partagent le même canal de communication et elles échangent entre elles des trames. Le protocole CSMA/CD a été défini pour l'accès partagé à ce média de communication.

Les premiers réseaux de type *Ethernet* utilisaient une technologie de type câble coaxial pour fonctionner selon une topologie en bus (cf. figure 2.3). Ces réseaux fonctionnaient sans équipement actif intermédiaire. L'évolution des technologies a permis d'abandonner ce type de technologie et de passer à des réseaux construits sur une topologie en étoile. Le cœur de ces réseaux est composé d'équipements actifs intelligents (par exemple pour une meilleure gestion du trafic). Nous étudierons plus précisément les réseaux Ethernet en étoile puisque ceux-ci nécessitent la présence d'équipements actifs pour fonctionner. En effet, la construction de la topologie en étoile nécessite du matériel actif pour connecter le cœur du réseau et les autres éléments (PC, Imprimantes, équipements actifs , etc..). La figure 2.4 présente un exemple de réseau typique sur lequel nous avons conduit nos expérimentations.

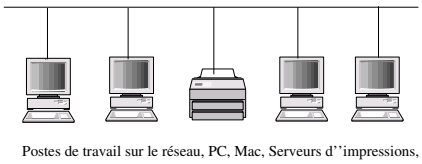

FIG. 2.3 – Exemple d'une topologie en bus

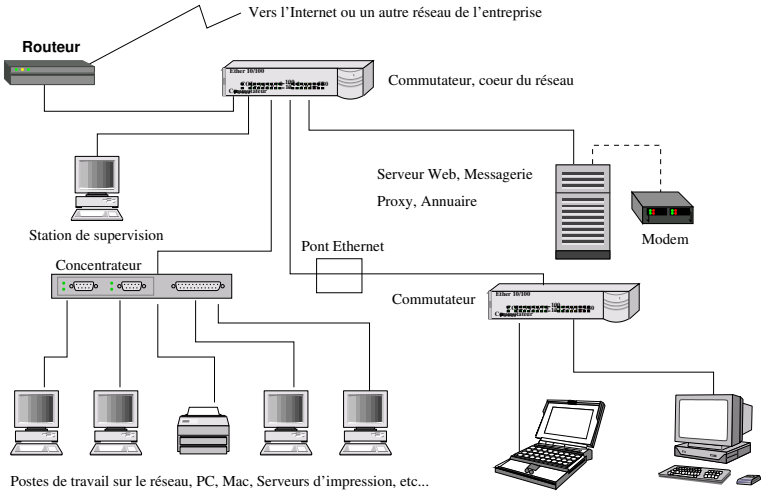

FIG. 2.4 – Exemple d'une topologie en étoile

Avec l'évolution des techniques relatives à la couche de liaison de données, plusieurs équipements actifs existent sur le marché informatique, dont principalement :

- les ponts Ethernet (Bridges) : ces équipements actifs ne font que répéter le signal du canal de communication à destination des autres interfaces qu'ils possèdent,
- les concentrateurs (Hubs) : ces équipements actifs répètent le signal reçu sur tous les ports de sortie excepté celui par lequel est arrivée l'information,
- les commutateurs (Switches) : ces équipements actifs répètent le signal sur le port de sortie qui correspond à l'adresse MAC (Medium Access Control) du destinataire ou sur tous les ports si le port de destination est inconnu (retransmission en mode de diffusion),
- les routeurs : ces équipements actifs transportent les paquets IP d'un réseau vers un autre réseau de destination.

L'avantage d'une topologie en étoile par rapport à une topologie en bus est que lorsque une connexion du réseau devient défaillante sauf sur le cœur du réseau, les communications sur le réseau perdurent à l'exception du brin défaillant, sauf si tous les services du réseau sont concernés par cette défaillance. Il s'agit là effectivement d'une meilleure sécurité pour assurer une qualité de service plus importante aux usagers.

2.3.3 La couche réseau

Cette couche permet de gérer la façon par laquelle les informations (ici paquets) sont acheminées depuis l'émetteur jusqu'au destinataire, et de s'assurer que l'information sera récupérée dans le même ordre qu'elle a été émise. Le protocole réseau le plus usité de nos jours reste le protocole IP [39], avec les deux couches de transport : UDP [96] et TCP [93]. Pour avoir un aperçu du fonctionnement d'un réseau, nous allons présenter quelques outils qui servent dans le suivi et l'observation de la couche réseau. Nous ne détaillerons pas les protocoles associés.

Ping-Pong Pour vérifier la connectivité entre deux ordinateurs possédant une adresse IP, le protocole ICMP (Internet Control Message Protocol) [38] est utilisé. Son fonctionnement de base est le suivant : émission d'un datagramme vers une machine de destination (ICMP Echo-Request), qui demande à la machine de destination de répondre par un datagramme vers l'émetteur (ICMP Echo-Reply). Son fonctionnement est relativement simple, mais l'utilisation de ce protocole permet de déterminer rapidement la panne d'un équipement actif du réseau. Il s'agit ici d'utiliser une des nombreuses fonctionnalités de ce protocole. La commande système associée la plus utilisée est la commande *ping* (cf. table 2.1 par exemple).

Ping de pacifique.iufm.unice.fr vers	www.unice.fr
Réponse de www.unice.fr	octets=32 temps <10 ms TTL=128

Tab. 2.1 – Exemple de résultat de la commande ping

La table 2.1 donne le compte rendu de l'exécution de la commande *ping*. La machine *www.unice.fr* a répondu avec un temps inférieur à 32 ms. Le paquet IP de retour a une durée de vie de 128 (Time To Live, le délai de vie du paquet sur le réseau IP).

Résolution des Adresses Quel que soit l'emplacement de la connexion physique des ordinateurs (Local Area Network (LAN) ou Wide Area Network (WAN)), un autre protocole entre en jeu afin d'effectuer le lien entre la couche réseau et la couche liaison de données. Ce protocole, appelé ARP (Address Resolution Protocol) [65] , permet de faire l'association de l'adresse IP de l'équipement avec l'adresse physique de la carte réseau de l'équipement. Si deux machines sont sur le même réseau local, le protocole ARP associera l'adresse IP de la machine avec l'adresse Ethernet de la carte réseau (cf. table 2.2). Ce protocole permettra à la couche réseau de joindre la machine destinataire et de lui transmettre la ou les trames de données. Si les deux machines ne sont pas sur le même réseau local (cas du WAN), un appel à ce protocole est également réalisé. La machine utilisera tout d'abord sa table de routage pour trouver le routeur intermédiaire,

recherchera son adresse Ethernet, et transmettra donc le message à ce routeur pour que celui-ci soit acheminé. La commande système qui est associée est la commande *arp*.

arp -a		
Adresse Internet (IP)	Adresse physique (MAC)	Méthode d'apprentissage
192.168.80.30	00 :04 :76 :97 :86 :46	dynamique

TAB. 2.2 – Exemple de résultat de la commande arp

La méthode d'apprentissage des adresses Ethernet par le protocole ARP est de deux types :

- Statique : les entrées dans cette table sont faites de manière manuelle (faites par l'administrateur)
- Dynamique : l'adresse Ethernet de l'hôte distant est apprise par le protocole, mise en cache pour être réutilisée ultérieurement. Il existe un délai de garde à partir duquel l'adresse sera effacée du cache (en général 5 minutes).

Suivi des trames IP Un autre outil qui utilise le protocole ICMP pour avoir le chemin parcouru par les paquets IP entre deux stations distinctes est *traceroute*. La table 2.3 donne le chemin parcouru par un paquet IP entre une station du réseau 134.59.80.0 et la station d'adresse 134.59.1.71 (www.unice.fr). Cet outil utilise la spécification de la durée de vie d'un paquet IP dans le réseau (champ TTL du paquet IP) pour découvrir le chemin parcouru entre l'émetteur et le destinataire.

traceroute	www.unice.fr
1 <10 ms <10 ms <10 ms	gw-80.iufm.unice.fr [134.59.80.254]
2 <10 ms <10 ms <10 ms	router-priv [172.16.250.1]
3 <10 ms <10 ms <10 ms	www.unice.fr [134.59.1.71]

TAB. 2.3 – Exemple de résultat de la commande traceroute

Les trois commandes présentées ci-dessus (*ping*, *arp* et *traceroute*) sont des commandes fournies en standard dans les systèmes d'exploitation. Ces commandes donnent un diagnostic primaire d'une perte de connectivité entre deux éléments du réseau (soit sur un LAN, voire même dans le cas d'un WAN). Toutefois, leurs possibilités limitées de diagnostic ont ouvert la voie vers des outils d'administration de systèmes et de réseaux par l'utilisation de protocoles d'administration de réseau comme décrit dans la section suivante.

Scan d'un réseau Un outil du domaine public, *nmap* [102], permet de scanner un réseau cible afin d'y trouver les services existants ainsi que de prédire le type de système d'exploitation par adresse IP découverte. Le résultat obtenu par cet outil de scan peut être utilisé pour définir de façon très approximative la topologie du réseau cible. En effet, les routeurs, les commutateurs sont détectés par cet outil, ce qui donne cette approximation.

Il fonctionne comme suit : par exemple, la commande `nmap -sT 192.168.80.0` cherchera tous les éléments du réseau et leurs services associés en partant de l'adresse IP `192.168.80.1` jusqu'à l'adresse `192.168.80.255`. Le résultat (voir un extrait table 2.4) de cet outil ne sera pas directement exploitable, mais celui-ci permet de connaître les éléments du réseau actif et leurs services.

nmap -sT 192.168.80.0		
Interesting ports on rtlocal (192.168.80.222) :		
Port	State	Service
79/tcp	open	finger
514/tcp	open	shell
2001/tcp	open	dc
6001/tcp	open	X11 :1
TCP Sequence Prediction : Class=random positive increments		
Difficulty=2198 (Medium)		
Remote operating system guess : Cisco Router/Switch with IOS 11.2		

TAB. 2.4 – Exemple de résultat de la commande *nmap -sT 192.168.80.0*

2.4 L'administration de réseau

Un outil typique destiné à l'administration de réseaux consiste en un outil qui fonctionne en mode Client/Serveur, associant une station d'administration de réseau (NMS) et les équipements actifs du réseau (cf. figure 2.5). Pour fonctionner, ces outils utilisent soit le protocole SNMP (voir section 2.4.1) dans le monde des réseaux IP ou le protocole CMIP (voir section 2.4.2) dans le monde des télécommunications. Nous parlerons plus précisément du protocole SNMP puisque notre travail de recherche s'est illustré plus spécifiquement dans le contexte de l'administration des réseaux IP et des réseaux locaux (LAN).

2.4.1 Le protocole d'administration de réseau : SNMP

SNMP (Simple Network Management Protocol) [86] est un protocole d'administration de réseau qui se veut simple dans son utilisation. Afin de pouvoir l'utili-

ser, il est nécessaire que les équipements actifs du réseau soient équipés d'un agent SNMP. Partant de cette hypothèse, il est possible de les interroger en utilisant le protocole SNMP. L'information qu'un administrateur réseau pourra recueillir dans un agent SNMP se trouve regroupée dans la MIB (Management Information Base) décrite ci-après. Ce protocole étant basé sur un modèle Client/Serveur (cf. figure 2.5), la station d'administration émet un paquet UDP représentant une requête SNMP à destination de l'équipement actif du réseau à administrer. Cette requête doit contenir le nom de la communauté SNMP de lecture (respectivement le nom de la communauté de lecture/écriture) pour accéder en lecture aux variables de la MIB SNMP (respectivement en lecture/écriture).

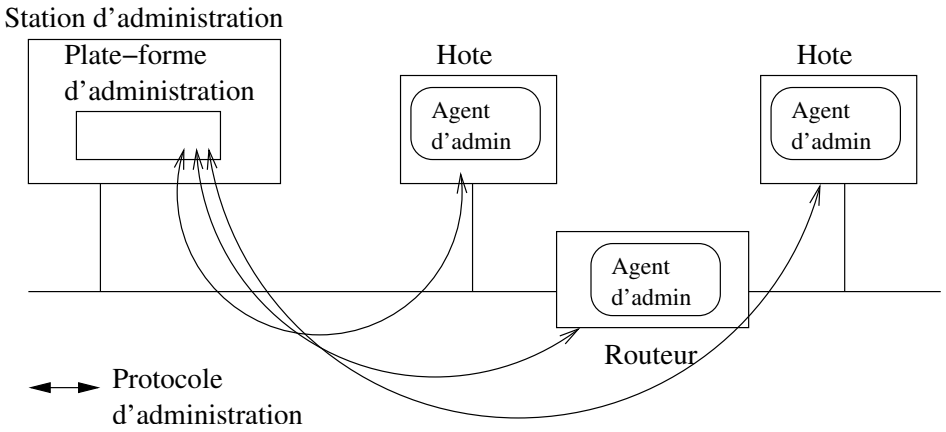

FIG. 2.5 – Architecture Client/Serveur sur laquelle se base l'utilisation d'un protocole d'administration dans un réseau

Lorsque l'agent SNMP est en mesure de répondre, il retourne, dans un paquet UDP vers l'émetteur, les valeurs des variables SNMP demandées.

2.4.1.1 MIB-2 SNMP

Chaque agent SNMP met à la disposition de la station d'administration des objets (appelés aussi variables) qui sont structurés dans une MIB (Management Information Base) [104] définie par la RFC-1213 [87]. Ces objets sont réunis dans des groupes (le groupe *system* décrit l'équipement actif, et par exemple la valeur de la variable *system.sysDescr.0* peut être la chaîne de caractères suivante : *Linux olddell.iufm.unice.fr 2.4.18-6mdk)*.

Chaque objet est identifié dans la MIB SNMP par un *oid* (Object IDentifier, voir la figure 2.6) qui définit sa position au sein de la structure arborescente de la MIB. En suivant l'arbre présenté par la figure 2.6 depuis la racine jusqu'à une feuille, on forme l'*oid*. Cet *oid* est constitué d'une suite d'étiquettes numériques séparées par des points.

Par exemple, l'*oid* 1.3.6.1.2.1.1.1.0 (cf. figure 2.7) correspond à la variable *systemDescr* du groupe *system* de la MIB. La valeur 0 à la fin de l'*oid* correspond

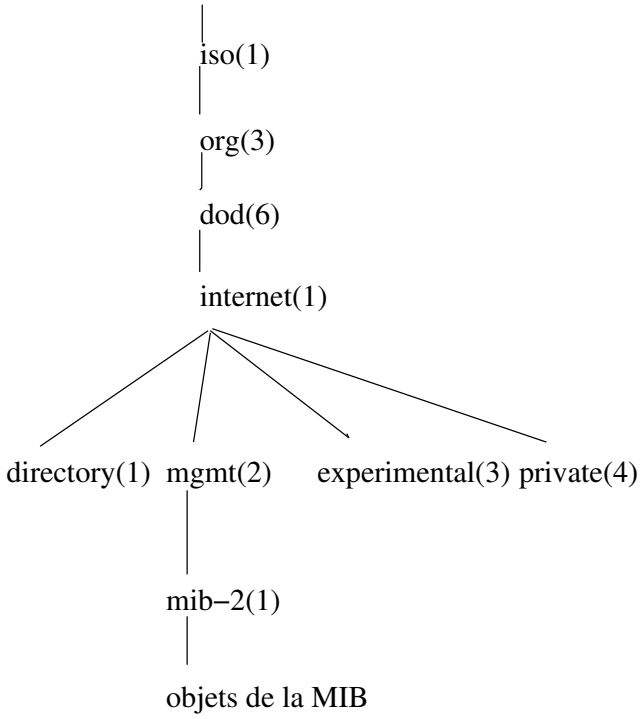

FIG. 2.6 – Structure d'indexation des données dans la MIB-2 SNMP

au fait que la variable recherchée est un objet simple (*system.systemDescr.0*). Par contre si la variable est une table on trouvera *oid.*<valeur index 1>....<valeur index n> (par exemple *interfaces.ifTable.ifEntry.1* jusqu'à la valeur *interfaces.ifTable.ifEntry.n*).

Les groupes les plus utilisés dans la MIB-2 SNMP sont regroupés dans la table 2.5 et une partie de la structure de la MIB SNMP est présentée par la figure 2.7.

Un exemple : le groupe IP de la MIB-2 SNMP Le protocole IP est un protocole qui délivre des datagrammes (mode non connecté) sur le réseau. Le groupe IP fournit les informations relatives au fonctionnement du protocole IP (oid de base du groupe IP = 1.3.6.1.2.1.4, c'est-à-dire iso.org.dod.internet.mgmt.mib-2.ip) sur l'équipement actif. On y trouve, par exemple, la table de routage (*ipRouteTable*), la table ARP par interfaces de l'équipement actif (*ipNetToMediaTable*) et une autre table qui donne les informations des adresses IP de l'équipement actif par numéro d'interface (*ipAddrTable*).

2.4.1.2 Fonctionnement du protocole

Le protocole SNMP définit 5 types de messages (Protocol Data Unit) (voir la table 2.6). Les messages de type *Get-Request*, *Get-Next-Request* sont des messages

Nom de la MIB-2 SNMP	Utilité
System	Nom, emplacement et description de l'équipement et notamment le type de service fourni par l'équipement (hub, switch, router,..)
Interfaces	Interfaces réseau et les données liées au trafic mesuré Par exemple : Type et nombre d'interfaces réseau
Ip	Statistiques sur le trafic IP, table de routage, table ARP, ... (...)
Icmp	Statistiques sur le trafic ICMP
Tcp	Paramètres et statistiques du trafic TCP, par exemple : la table des connexions TCP
Udp	Paramètres et statistiques du trafic UDP
Egp	Informations sur la mise en œuvre du protocole EGP (External Gateway Protocol)
Transmission	Informations au sujet des moyens de transmission et des protocoles d'accès aux interfaces de l'équipement
Snmp	Paramètres et statistiques du trafic SNMP

Tab. 2.5 – Informations principales de la MIB-2 SNMP

utilisés pour de la collecte d'informations d'administration sur un agent SNMP.

Le message de type *Get-Request* est émis depuis un outil de supervision vers un agent SNMP. L'agent SNMP répond par un message de type *Get-Response* contenant la valeur des variables demandées.

Le message particulier *Get-Next-Request* est un message qui demande à l'agent SNMP la variable suivante dans la MIB SNMP. Par exemple, un message *Get-Next-Request* 1.3.6.1.2.1.1.1.0 (*system.systemDescr.0*) vers un agent SNMP sera suivi d'un message de type *Get-Response* contenant la valeur de l'objet d'*Oid* 1.3.6.1.2.1.1.2.0 correspondant à l'objet *system.sysObjectID*. Par un échange de messages de type *Get-Next-Request* et *Get-Response* entre l'outil de supervision et un agent SNMP, il est possible d'obtenir toute la MIB de l'agent SNMP.

Un message de type *Set-Request* émis par l'outil de supervision à destination de l'agent SNMP permet la mise-à-jour d'une ou plusieurs variables. La réponse en retour est soit la liste des variables mises à jour soit un message d'erreur rapportant une incohérence dans la mise à jour (cf. table 2.9).

Un message de type *Trap* peut être émis depuis l'équipement actif vers la station d'administration, message non sollicité, afin de prévenir la station d'administration d'un événement pour lequel l'agent SNMP a été programmé. Par exemple, une *trap* SNMP peut être émise lors du changement d'état d'une interface réseau (ex : passage du statut *linkUp* au statut *linkDown*). L'administrateur configure l'agent SNMP pour que celui-ci émette une alerte, en spécifiant le type

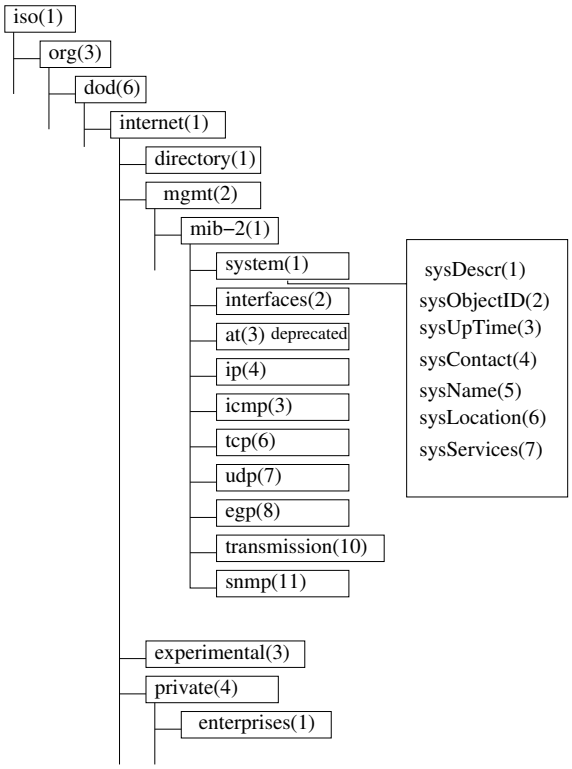

FIG. 2.7 – Les groupes d'objets de la MIB-2

d'alerte et l'adresse IP de la station d'administration de réseau.

2.4.1.3 Quelques exemples d'informations stockées dans la MIB-2 SNMP

Les commandes que nous avons présentées dans la section 2.3.3 permettent d'obtenir des informations sur le fonctionnement du réseau, mais ces mêmes informations peuvent être obtenues en collectant des données dans la MIB d'un agent SNMP. Pour cela, nous allons présenter le groupe *system*, et quelques tables du groupe *ip* donnant des informations sur la table ARP et la table de routage de l'élément.

2.4.1.4 Groupe *system*

Pour déterminer les types de services fournis par l'équipement actif sur lequel nous collectons des données, nous utilisons les informations contenues dans le groupe *System.sysServices* de la MIB.

La liste ci-dessous donne un exemple des valeurs de cette variable en fonction des services fournis par un équipement actif. Le niveau de fonctionnalité est :

– 1 physical (par exemple répéteur ou concentrateur)

Emetteur	Message	Fonctionnalité
Station	GetRequest	obtenir une variable
Station	GetNextRequest	obtenir variable suivante (si existante, sinon retour d'erreur)
Station	SetRequest	modifier la valeur d'une variable
Agent SNMP	GetResponse	retour de la réponse
Agent SNMP	Trap	message d'alerte de l'agent SNMP vers la station d'administration

TAB. 2.6 – Opérations définies par le protocole SNMP

- 2 datalink/subnetwork (par exemple commutateur)
- 3 internet (par exemple passerelle IP, routeur)
- 4 end-to-end (par exemple élément du réseau, PC)
- 7 application (par exemple un relais de messagerie, une imprimante en réseau)

Par exemple, pour un concentrateur exécutant des fonctions de répéteurs avec un agent SNMP, la valeur de la variable *system.sysServices* sera 1. Un élément n'effectuant que des tâches de routage de trafic aura comme valeur de la variable *system.sysServices* la valeur 4. Un élément effectuant des fonctions de niveau 2 et de niveau 3 aura comme valeur pour la variable *system.sysServices* la valeur 6. La technique pour connaître le type de service fourni est d'utiliser la décomposition de la valeur de la variable suivant la somme de $2^{Niveau-1}$, c'est-à-dire l'utilisation d'un masque binaire, comme définie dans la RFC-1213 [87], où chaque niveau correspond à la couche OSI.

2.4.1.5 Groupe *ip* : la table de routage

La table de routage d'un réseau local nous donne le chemin parcouru par les paquets IP entre le réseau local et le réseau extérieur. Pour obtenir ces informations par le biais de l'agent SNMP, nous collectons la table *ip.ipRouteTable*. Chaque ligne de la table 2.7 est une entrée de la table *ip.ipRouteTable*.

2.4.1.6 Groupe *ip* : la table *ARP*

Sur les équipements actifs du réseau local (par exemple un routeur ou un commutateur), une collecte de la table ip.ipNetToMediaTable permet d'avoir les mêmes informations que celles obtenues par la commande Unix *arp -a* (voir section 2.3.3). La table 2.8 en donne un aperçu.

SNMP : IpRouteTable			
ipRouteDest	ipRouteNextHop	ipRouteMask	ipRouteIfIndex
255.255.255.255	0.0.0.0	255.255.255.255	2
192.168.81.0	192.168.80.253	255.255.255.0	2
192.168.80.0	0.0.0.0	255.255.255.0	2
192.168.1.0	192.168.80.244	255.255.255.0	2
192.168.82.0	192.168.80.253	255.255.255.0	2
192.168.11.0	192.168.80.244	255.255.255.0	2
192.168.10.0	192.168.80.244	255.255.255.0	2
127.0.0.0	0.0.0.0	255.0.0.0	2
0.0.0.0	192.168.80.253	0.0.0.0	2

TAB. 2.7 – Exemple d'une table de routage obtenue en SNMP

Address IP (ipNetToMediaNetAddress)	Address Physique (ipNetToMediaPhysAddress)	Interface (ipNetToMediaIfIndex)
192.168.80.30	00 :04 :76 :97 :86 :46	2
192.168.80.241	00 :06 :5B :D1 :50 :5B	2
192.168.80.10	00 :30 :05 :08 :23 :39	2
192.168.80.245	00 :02 :16 :57 :43 :00	2
192.168.80.244	00 :10 :7B :3A :20 :B5	2
192.168.80.7	00 :D0 :B7 :7E :73 :AD	2

TAB. 2.8 – Exemple d'une table ARP obtenue en SNMP

2.4.1.7 Gestion des interfaces

Nous présentons ici la commande Unix *snmpset* qui permet de modifier une valeur dans la MIB-2 SNMP. L'exemple de la table 2.9 permet de changer le statut d'une interface d'un commutateur de Up (valeur 1) à Down (valeur 2). Le protocole SNMP retourne la valeur qui a été affectée par l'opération *SET* du protocole.

Commande	snmpset -v1 -c private 192.168.80.246 interfaces.ifTable.ifEntry.ifAdminStatus.4 integer 2 (down)
Résultat	interfaces.ifTable.ifEntry.ifAdminStatus.4 = down(2)
Commande	snmpset -v1 -c private 192.168.80.246 interfaces.ifTable.ifEntry.ifAdminStatus.4 integer 4 (Valeur incorrecte)
Résultat	Error in packet. Reason : (badValue) The value given has the wrong type or length.

TAB. 2.9 – Exemple d'utilisation de la commande snmpset sur un commutateur

2.4.1.8 Evolution du protocole SNMP

Le nouveau standard : le SNMP V3 Les fonctionnalités de base du protocole SNMP V1 (mais aussi SNMP V2, non détaillé ici, voir [19]) que nous venons de décrire, n'offrent pas de mécanisme de sécurité permettant d'authentifier la source d'un message, ni de fournir un quelconque cryptage des données. Pour cela, le protocole SNMP V3 [20] a été défini pour palier aux insuffisances des deux versions précédentes du protocole.

Ce protocole vise essentiellement à inclure la sécurité des transactions, notamment l'identification des parties et la confidentialité. Le protocole SNMP V3 se veut modulaire dans son implémentation :

- Transport : module responsable du transport. Il se présente sous la forme d'un module et non pas dans le moteur SNMP pour rendre indépendante la description SNMP V3 de UDP.
- Module de traitement : module responsable d'encoder et décoder les paquets SNMP en respectant le numéro de version du standard (V1, V2 ou V3). On a fait de cette tâche un module externe au moteur pour permettre l'adaptation aux prochaines versions de SNMP. Plusieurs modules de traitements peuvent fonctionner simultanément. Il est donc possible pour un moteur SNMP de traiter des messages de différentes versions en même temps.
- Sécurité : la sécurité concerne l'authentification, l'encryption et la vérification du temps écoulé entre l'échange de deux paquets afin que la capture d'un paquet ne permette pas son utilisation ultérieure à des fins de deni de service.

- Applications : il s'agit des processus qui interagissent avec le moteur SNMP en utilisant des messages qui peuvent être définis dans le protocole ou des messages décrits par une mise en œuvre spécifique du moteur. Les applications sont développées pour effectuer des opérations de gestion spécifiques. Les objectifs peuvent être très variés d'une application à une autre mais elles utilisent en commun le même moteur SNMP pour effectuer les opérations de gestion. Selon les applications, l'entité peut jouer le rôle de superviseur et converser avec un autre superviseur en échangeant des requêtes d'informations. Cela permet un échange d'informations sur le système d'administration qui n'existait pas dans les versions précédentes du protocole.
- Contrôle d'accès : le module du contrôle d'accès doit décider si une requête est permise et si une réponse peut être envoyée ou l'ignorer si une personne non autorisée a fait la requête.

La sécurité dans le protocole SNMP V3 peut aller de l'authentification d'un utilisateur (plus sécurisé que le nom de la communauté SNMP en V1 et V2), à l'authentification de l'émetteur des messages et au cryptage des paquets échangés. Les aspects liés à la sécurité du protocole SNMP V3, contenues dans le *User Security Module* sont décrites ci-dessous.

Authentification : l'authentification a pour but d'assurer que le paquet n'a pas subi de modification en cours de route et que le mot de passe de l'entité émettrice est valide. Elle utilise les fonctions de hachage à une direction (MD5 et SHA-1). Les données obtenues et le code de hachage sont transmis sur le réseau. Avec cette technique, le mot de passe est validé sans qu'il ait été transmis sur le réseau.

Localisation des mots de passe : ce mécanisme permet que la sécurité d'un domaine d'administration soit garantie même si la sécurité d'un des agents SNMP du domaine a été compromise. SNMPv3 préconise l'utilisation d'un seul mot de passe, pour tous les échanges avec les agents SNMP, mais ajoute une étape de localisation. Un mot de passe localisé ne fonctionne qu'avec un seul agent. Pour localiser, une chaîne de caractères unique est déterminée pour chaque agent. SNMPv3 utilise le *ContextEngineID*[1]. Cette chaîne de caractères est générée par un ensemble de données relatif à l'agent : adresse MAC de la carte ethernet, adresse IP, etc.. Le *ContextEngineID* est groupé avec le mot de passe et le résultat est transmis à une fonction de hachage à une direction où il sera vérifié comme dans l'étape d'authentification.

Encryption : l'encryption empêche de lire les informations de gestion contenues dans un paquet SNMPv3. SNMPv3 utilise pour l'encryption un deuxième

[1]Un agent SNMP V3 peut contenir plusieurs agents SNMP, cette chaîne permet donc de converser avec le bon agent SNMP

mot de passe partagé entre la station de gestion et l'agent. SNMPv3 se base sur DES (Data Encryption Standard) pour effectuer l'encryption.

Estampillage du temps : ce mécanisme vise à empêcher la réutilisation d'un paquet SNMPv3 valide déjà transmis. Le temps est estampillé sur chaque paquet. Quand le paquet est reçu, le temps actuel est comparé avec le temps indiqué sur le paquet. Si la différence est supérieure à 150 secondes, le paquet est ignoré.

2.4.2 Le protocole CMIP

Le protocole CMIP [104] est un protocole d'administration de réseau, qui supporte l'échange d'informations entre l'application d'administration et les agents d'administration.

SNMP a été développé comme solution de transition plus simple de CMIP/CMIS en attendant que le support logiciel de CMIP/CMIS soit suffisant. SNMP avait pour objectif de disparaître dès la généralisation de CMIP/CMIS.

L'information d'administration échangée entre l'application et les agents est faite par le biais d'objets (CMIP/CMIS), qui représentent l'entité à administrer[2]. Ces objets peuvent être modifiés ou contrôlés et ils peuvent être utilisés pour effectuer des tâches sur l'agent administré.

Le protocole CMIP est un protocole largement utilisé dans le domaine des télécommunications. Cependant, ce protocole consomme beaucoup de ressources sur le système administré, ce qui fait que ce protocole n'est pas très largement diffusé. De plus, le protocole CMIP est défini pour fonctionner sur la pile du protocole OSI. Cependant, le standard utilisé de nos jours dans la majorité des environnements de réseaux locaux (LAN) est le protocole TCP/IP[3]. Pour preuve du succès plus important de SNMP, c'est que la plupart des équipements actifs n'implémentent que le protocole SNMP [4].

2.4.3 Administration répartie

Ce que nous venons de décrire, aussi bien en terme d'outils permettant la détection des incidents du réseau, ou en terme d'utilisation d'un protocole d'administration de réseau (SNMP) sont des techniques centralisées. Nous allons maintenant présenter quelques évolutions vers l'administration de réseau distribuée, en présentant l'évolution des MIBs SNMP pour intégrer de l'administration de réseau répartie, proposée par le groupe de l'IETF DISMAN (pour Distributed

[2]L'objet est l'encapsulation des données qui sont associées à la ressource, combiné à la partie logicielle adaptée à la ressource, qui implémente les opérations qui peuvent être faites sur les données.

[3]TCP/IP ne calque pas complètement sur les couches OSI.

[4]Une implémentation de CMIP sur TCP/IP existe et elle s'appelle CMOT [24] : CMIP over TCP.

Management). L'engorgement de plus en plus important des stations d'administration a incité les fondateurs à proposer une amélioration de la MIB SNMP permettant une administration répartie. Deux propositions ont été faites, l'approche par scripts et l'approche par MIB.

Dans cette extension de SNMP, on définit les tâches du gestionnaire ainsi que les tâches qui sont déléguées par tout gestionnaire au gestionnaire de niveau intermédiaire. Le choix des gestionnaires intermédiaires se base sur la notion de proximité (voir figure 2.8), c'est-à-dire au plus proche des équipements actifs du réseau.

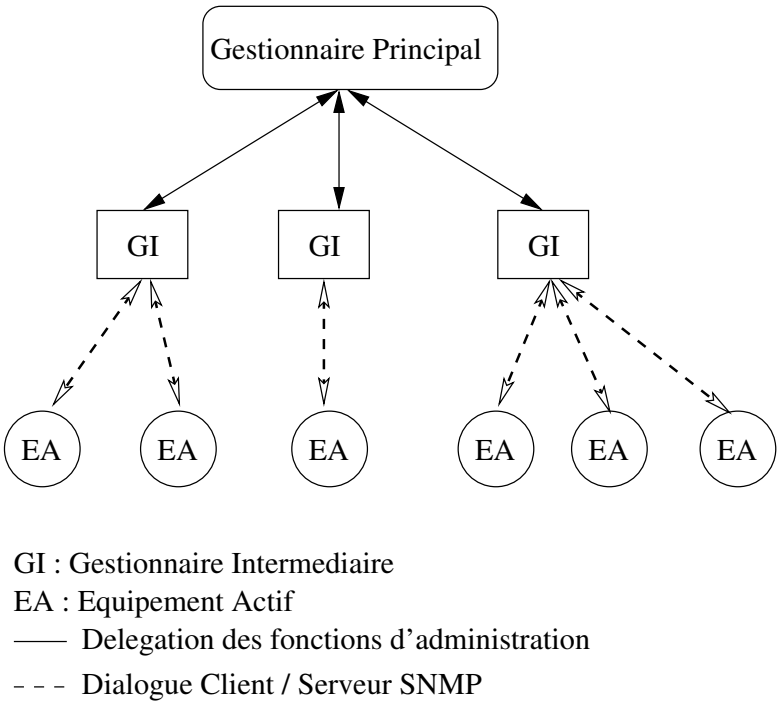

FIG. 2.8 – Délégation à un gestionnaire intermédiaire des opérations d'administration

2.4.3.1 Approche par Script

L'approche par Scripts est de permettre la mise en œuvre d'opérations d'administration les plus proches possible des équipements actifs du réseau, face à l'augmentation croissante du nombre d'éléments à administrer. Les deux MIBs décrites ci-dessous peuvent travailler de concert afin de réaliser automatiquement des tâches d'administration.

MIB *Script* Une telle solution, comme décrite par Pras dans [66], fonctionne selon un principe de délégation des méthodes d'administration. Le fonctionnement est relativement simple : la station d'administration du réseau transfert

les scripts vers les stations d'administration intermédiaires afin que celles-ci exécutent le code des scripts [80]. Plusieurs langages peuvent être utilisés pour l'exécution des scripts, comme tcl/tck, shells Unix, etc...

MIB *schedule* Il s'agit de positionner une valeur donnée à une période définie dans un objet de la MIB. La période peut être unique, dans le sens où la valeur sera positionnée une seule fois, périodique ou datée. En utilisant cette MIB [45], certaines tâches très périodiques que doit exécuter un administrateur peuvent être programmées et l'équipement actif a la responsabilité d'effectuer les modifications demandées. Par exemple, ouvrir administrativement certains accès réseau en changeant la valeur du statut d'administration d'une interface réseau (`interfaces.ifTable.ifEntry.ifAdminStatus`).

2.4.3.2 Approche par MIB

Nous allons détailler les MIBs SNMP qui ont été définies par le groupe de travail du DISMAN pour distribuer les opérations d'administration dans les gestionnaires intermédiaires du réseau.

Les opérations SNMP déportées L'utilisation de la MIB DISMAN *remote operation* [100] permet au gestionnaire de lancer à distance les programmes *ping*, *traceroute* et *lookup*[5] depuis un gestionnaire intermédiaire. Le résultat de ces diverses commandes est enregistré dans des tables, chacune étant dédiée à une commande. Il est possible d'avoir simultanément le résultat des trois commandes en consultant les tables associées depuis la station de supervision.

MIB log notifications Il s'agit de la définition d'une MIB qui permet de sauvegarder les alertes émises par les autres éléments du réseau évitant ainsi toute perte sur la station d'administration [43]. L'enregistrement des événements peut être associé à la mise en place d'un filtre qui permet de n'enregistrer que les événements importants. Les informations concernant les événements qui sont enregistrés sont : l'*oid* de l'objet qui a généré cette alerte, et l'adresse IP de l'émetteur. Toutefois, s'il s'agit du protocole SNMP V3, en plus des informations précédentes, l'entité et le nom du contexte sont enregistrés. Une donnée supplémentaire correspondant à l'heure d'enregistrement de l'alerte est associée afin que la station d'administration puisse reconstituer le déroulement des alertes. Par exemple, la notification du changement du statut d'une interface (*link-up* ou *link-down*) peut être enregistrée dans cette MIB, ce qui peut être d'autant plus pertinent si l'équipement actif se trouve sur un réseau distant.

[5] La commande *lookup* permet de connaître les informations de nommages des éléments dans le réseau, comme explicité par les commandes système du type *nslookup* [26].

MIB event Cette MIB permet de mettre en œuvre un mécanisme de déclenchement selon le résultat de l'évaluation d'une condition [41]. Cette MIB est programmée à l'avance depuis la station d'administration, et la surveillance de l'évaluation des conditions programmées permet le déclenchement automatique par l'agent SNMP de l'événement associé.

MIB *expression* Il s'agit de faire évaluer en utilisant cette MIB [42] des expressions selon des critères définis par l'administrateur du réseau. Le résultat de l'évaluation est enregistré dans une table de cette MIB qui peut être consulté a posteriori par l'administrateur. L'évaluation se fait au moment où l'administrateur fait une requête de lecture de l'expression. Celle-ci est automatiquement évaluée et le résultat stocké dans la table associée.

2.4.3.3 SNMP centralisé versus DISMAN

Les MIBs définies par le groupe DISMAN de l'IETF donnent une approche différente de l'administration réseau traditionnelle, où toutes les données sont analysées par une station centralisée. Il y a une utilisation des possibilités de calcul des équipements actifs du réseau, ce qui permet de déléguer certaines tâches d'administration. Cependant, l'utilisation des MIBs succinctement décrites dans cette section est consommatrice en terme de ressources de calcul et de stockage des résultats. Toutefois, cette approche va dans le sens d'une plus grande autonomie de gestion au niveau des équipements actifs.

2.5 Quelques outils incontournables pour l'administration

Il existe sur le marché différents types d'outils pour effectuer des tâches d'administration de systèmes et de réseaux. Nous décrirons les outils minimaux qui sont utilisés pour visualiser le trafic qui circule sur les réseaux dans la section 2.5.1. Une présentation succincte d'un outil commercial sera faite dans la section 2.5.2, et nous terminerons en présentant les autres méthodes pour effectuer de l'administration d'équipements actifs.

2.5.1 Des outils élémentaires

Il existe un nombre important d'outils destinés à l'administration des réseaux. Les plus simples d'utilisation d'entre eux sont des outils du domaine public. Par contre ces outils ne sont pas conçus pour être intégrés les uns avec les autres. Nous pouvons citer, à titre d'exemple, les outils suivants :

– *tcpdump* : trace des trames Ethernet que reçoit l'interface réseau du système

- *MultiRouter Traffic Grapher* [56] : permet d'obtenir des graphiques sur l'activité des équipements actifs du réseau
- *etherape* [27] : est une visualisation de ce que l'on peut obtenir par tcpdump (cf. figure 2.9)
- etc..

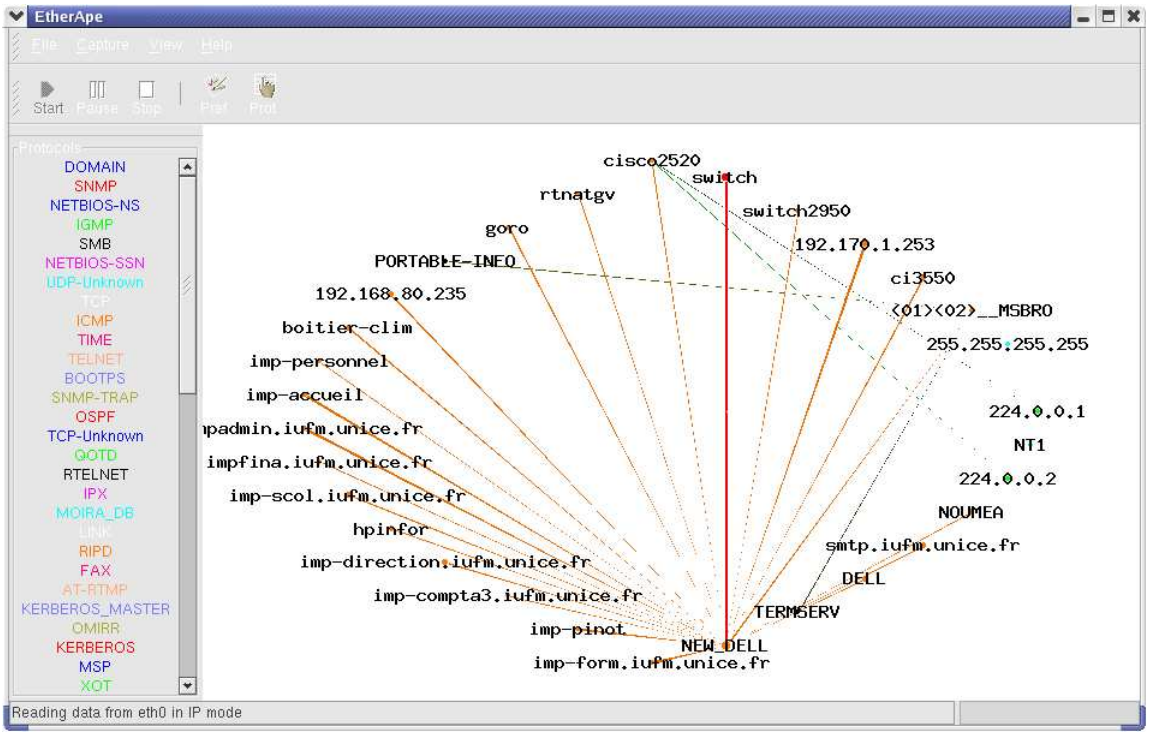

FIG. 2.9 – Plate-forme de supervision du trafic réseau instantané

Un nombre important de tels outils existent effectivement, mais ces outils présentent les mêmes lacunes d'intégration que ceux cités ci-dessus, c'est-à-dire qu'il est impossible d'utiliser les informations d'un outil automatiquement dans un autre outil par manque de standard dans les échanges d'information.

2.5.2 Plates-formes d'administration centralisée

2.5.2.1 Quelques exemples de plates-formes

Il se trouve que les plates-formes en exploitation sont des outils commerciaux. Les solutions proposées comme Network Node Manager [35], Tivoli [95] ou Cisco Works [23] sont des outils très performants pour effectuer le travail de supervision et d'administration de réseau. Ces outils propriétaires ont des possibilités étendues concernant la gestion de leur propre matériel (ex : Hp Network Node Manager avec des imprimantes réseaux HP), mais nettement moins concernant les équipements actifs des autres constructeurs.

Ces outils commerciaux permettent de suivre en temps réel l'activité des équipements actifs qui composent les réseaux de l'entreprise. Ces outils offrent la possibilité d'avoir la topologie du réseau (cf. figure 2.11), les liens entre les différents équipements composant le réseau de l'entreprise. Ces outils offrent aux administrateurs les possibilités de programmation des alertes, la gestion des fautes qui peuvent survenir au sein de l'infrastructure du réseau.

2.5.2.2 Architecture des plates-formes d'administration de réseau centralisée

La figure 2.10 présente le schéma traditionnel d'une plate-forme d'administration de réseau, avec d'un côté la station d'administration qui supporte les fonctionnalités d'administration : enregistrement des informations liées aux équipements actifs, protocole d'interrogation des équipements actifs, interface d'administration (GUI), noyau de l'application d'administration, c'est-à-dire tous les outils permettant à l'administrateur d'être au courant du fonctionnement du système.

De l'autre côté, on retrouve les agents d'administration (agents SNMP) avec leurs différentes implémentations de la MIB SNMP en accord avec leur fonctionnalité (routeur, commutateur, concentrateur, etc...)

La station d'administration gère la communication avec tous les équipements actifs (selon le protocole d'administration choisi, en général SNMP) et fournit le support d'une décision centralisée quant à l'administration du réseau. Une telle approche n'est plus forcement utilisable dans les réseaux actuels, à cause de l'augmentation de la taille des réseaux d'une part, et de leur organisation de plus en plus complexe, comprenant des sous-parties plus autonomes. Une administration répartie semble mieux passer à l'échelle, est plus robuste et plus réactive sur le temps d'exécution des fonctions d'administration [12, 49]. En effet, le nombre d'équipements actifs devenant important, la station d'administration centralisée devient donc le point faible de tout le système d'administration, tant par la charge réseau induite que par la quantité d'informations à collecter et à mémoriser. Il convient de noter que l'administrateur du réseau doit se trouver physiquement sur la station d'administration afin d'être en mesure de traiter les incidents provenant du réseau.

A titre d'exemple nous allons présenter `HP Network Node Manager` car c'est un outil complet et représentatif des plates-formes d'administration de réseau. Cette plate-forme permet à plusieurs administrateurs de se connecter à distance, via un navigateur Internet, sur la plate-forme d'administration. Ces administrateurs suivent donc les alertes et les incidents qui peuvent survenir sur le réseau à partir d'un ordinateur distant.

Afin d'avoir un aperçu général de la plate-forme `HP Network Node Manager`, nous avons installé et fait tourner cette plate-forme sur un réseau de production (sur le réseau illustré par la figure 2.4). Nous récupérons donc de la plate-forme

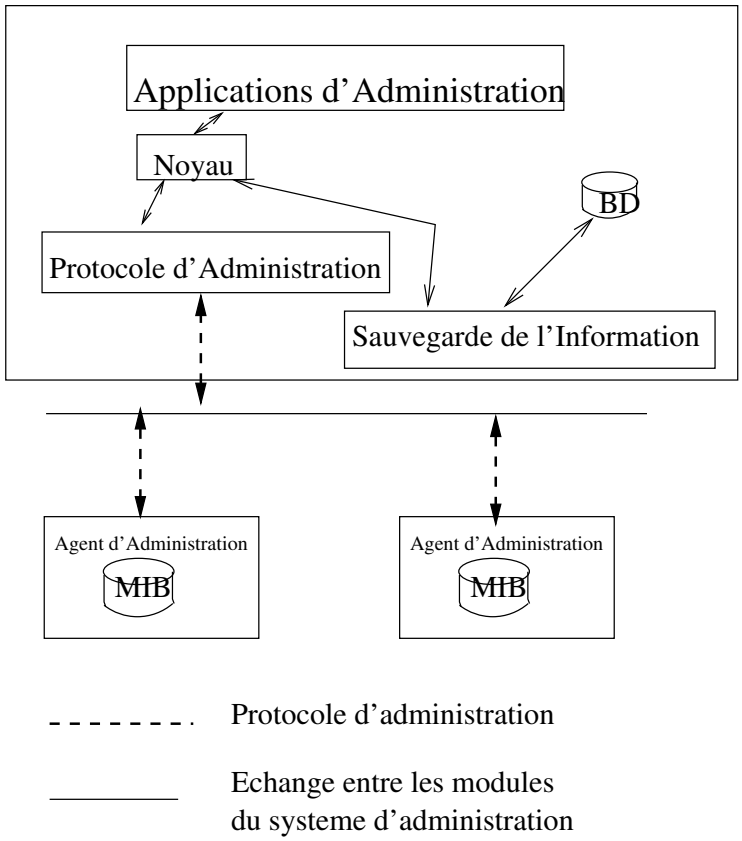

FIG. 2.10 – Plate-forme d'administration de réseaux centralisée

des informations visuelles de ce réseau.

La figure 2.11 présente une capture d'écran de la topologie du réseau présentée sur la figure 2.4. On y trouve les différents segments détectés par l'outil d'administration connectés sur le commutateur central du réseau local. Chaque segment est localisé par rapport à l'interface du commutateur qui le relie au reste du réseau (par exemple l'interface FastEthernet 0/9 pour le segment 8) et on trouve sur la figure les équipements actifs du réseau facilement détectables par un tel système d'administration : les routeurs et les commutateurs. Ils sont représentés par des figures en losange indiquant le fait qu'ils participent à la gestion du trafic du ou des segments du réseau. Sur la figure 2.11 présentée, deux routeurs Cisco ont été détectés (cisco2520 et gw-80) et ceux-ci sont connectés sur le commutateur central. Les brins qui relient les différents segments au commutateur central permettent d'avoir leur inter-connexion sur les interfaces du commutateur central (ici sur les interfaces de type FastEthernet : Fa0/9 et Fa0/24).

Afin d'avoir une vue plus détaillée d'un des segments du réseau, il suffit de le sélectionner dans la GUI de l'outil. Ansi la figure 2.12 montre les éléments découverts appartenant au segment 8 (PC, imprimantes, routeur Cisco 2520 et évidemment le commutateur central).

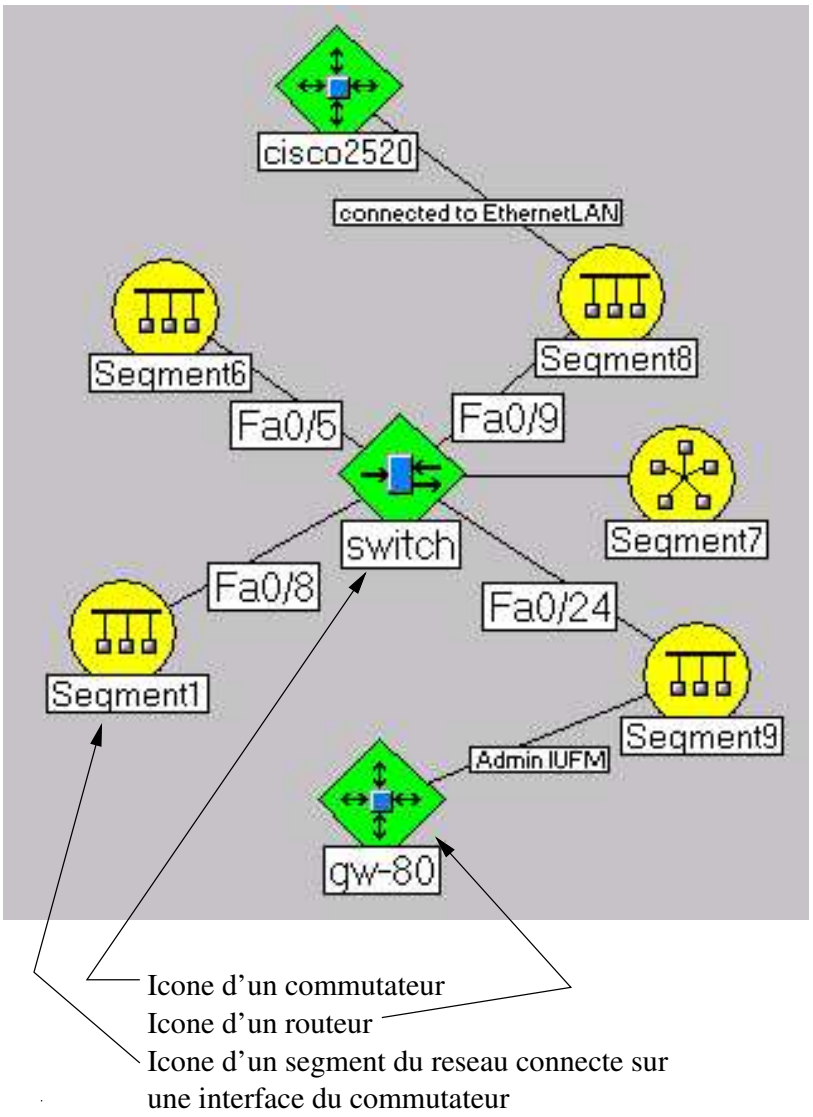

FIG. 2.11 – Capture d'écran de la topologie réseau obtenue par la plate-forme HP Network Node Manager

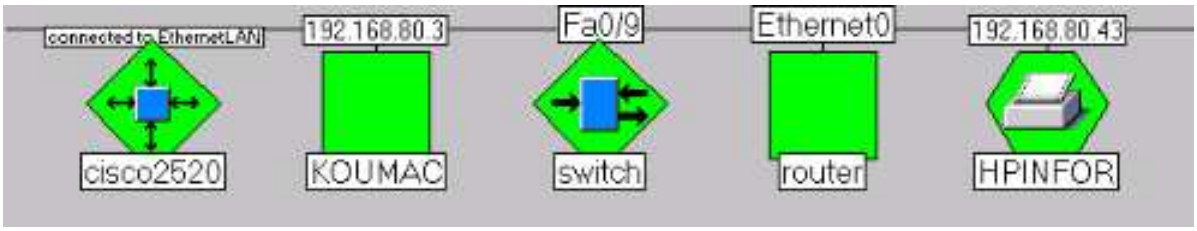

FIG. 2.12 – Capture du segment 8

2.5.3 Des plates-formes d'administration basées sur le Web

Étant donné que le protocole HTTP est devenu omniprésent, les constructeurs ont intégré un serveur HTTP au sein des équipements actifs du réseau. Les fonctionnalités fournies sont celles que le constructeur a bien voulu offrir, mais elles ont l'avantage de permettre une administration des équipements actifs via un navigateur. Cette technique d'administration à distance s'appelle le *Web-Based Management* [99]. La méthode utilisée pour la configuration se déroule entre le navigateur du poste Client et le Serveur de l'élément du réseau de la manière suivante :

- Le client crée une connexion TCP avec le serveur,
- Le client transmet une requête HTTP, typiquement un GET ou un POST au serveur,
- Le serveur retourne une entête HTTP suivie, éventuellement, d'une *applet* Java qui sert à établir le lien entre le poste client et le poste serveur,
- Le client rompt la connexion HTTP et l'*applet Java* prend le relais pour transmettre les ordres d'administration à l'équipement actif.

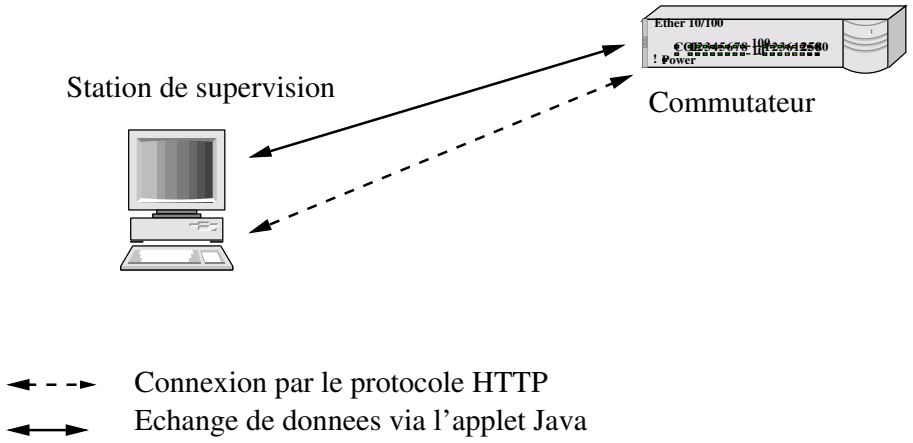

FIG. 2.13 – Principe du Web-Based Management

Toutefois cette méthode est une solution qui ne permet d'administrer les équipements actifs que un par un, alors que les réseaux sont composés de dizaines d'équipements actifs et qu'il faut parfois s'occuper de plus d'un équipement actif simultanément.

2.6 Bilan

Dans ce chapitre, nous avons détaillé les outils qui permettent d'effectuer des opérations simples d'administration, comme en utilisant les commandes Unix *ping*, *arp*, *traceroute*, etc.., commandes qui donnent des informations sensibles à l'administrateur du réseau sur la connectivité des réseaux et des éléments entre

eux. Pour obtenir de plus amples informations, un administrateur peut utiliser le protocole SNMP (ou HTTP) pour communiquer avec des équipements actifs qui composent le réseau qu'il gère, afin de prendre des décisions efficaces quant à la mise en œuvre de services du réseau. Des outils plus ou moins intégrés dans des plates-formes existent pour l'aider dans son métier, que ceux-ci soient libres de droits ou commerciaux.

Cependant la difficulté voire l'impossibilité d'étendre les fonctionnalités offertes par les plates-formes d'administration et notamment les commerciales, justifie d'envisager d'autres solutions. Les plates-formes à agents mobiles présentent en général la propriété de permettre à l'administrateur de programmer de nouvelles fonctions. Par ailleurs ces plates-formes offrant le concept d'agents mobiles permettent d'introduire plus d'autonomie dans la réalisation des fonctions d'administration et donc de décharger d'autant le travail de l'administrateur.

Le chapitre 3 présente l'état de l'art des plates-formes à agents mobiles qui sont utilisées dans ce contexte.

Chapitre 3

Les plates-formes à agents mobiles pour l'administration système et réseau : État de l'art

3.1 Introduction

Depuis quelques années, la communauté d'administration de systèmes et de réseaux a reconnu le fort potentiel qu'offrent les systèmes basés sur des agents et en particulier, sur des agents mobiles, pour effectuer des tâches d'administration [11] : décentralisation du contrôle, répartition des tâches de calculs liées à l'administration, autonomie, possiblilité de gain en bande passante, etc.

La popularité du langage Java [51] et en particulier, le fait qu'il masque l'hétérogénéité des systèmes, qu'il permet aisément la mobilité de code et qu'il propose des outils permettant d'appréhender les problèmes relevant de la sécurité, a plus précisément focalisé l'intérêt de cette communauté sur des systèmes à agents mobiles écrits en Java. Dans le contexte de l'intégration des agents mobiles pour l'administration système et réseau, nous avons étudié les plates-formes qui sont bâties au dessus d'une architecture en Java. Les plates-formes à agents mobiles sélectionnées, dans ce chapitre, ont pour objectif de fournir des outils d'administration de systèmes et de réseaux par le biais des agents mobiles. Pour utiliser ces agents mobiles dans ce contexte, il est souhaitable d'offrir la notion d'itinéraires construits et obtenus dynamiquement afin d'arriver à un degré élevé d'automatisation des tâches. Cependant, dans la majorité des plates-formes étudiées, les itinéraires sont obtenus de façon statique (via par exemple un fichier), et ce sera l'objet principal de ce travail de thèse de lever ce type de contraintes. Comme pour construire des itinéraires dynamiques il faut une connaissance tant qu'à faire automatique du réseau à administrer, nous étudierons dans ce chapitre l'état de l'art concernant un autre aspect : la construction dynamique de topologie du réseau.

En effet, les améliorations fournies par les équipementiers et la diminution des coûts des matériels actifs des réseaux (commutateurs, routeurs, concentrateurs équipés d'agents SNMP) permettent désormais de collecter de plus en plus d'informations du réseau. Il n'est plus nécessaire de collecter les informations de trafic directement sur les éléments (PC, Serveurs, etc..) car le cœur du réseau offre à son niveau l'équivalent. Il devient donc possible d'interroger uniquement les équipements actifs le constituant afin d'avoir une vue très fine de l'état général du réseau. Grâce à ces techniques et aux services inclus dans les équipements actifs, il a de même été possible de concevoir des algorithmes permettant la détermination fine de la topologie du réseau, c'est-à-dire de localiser le plus précisément possible les éléments (PC, Serveurs, etc..), le reste de l'architecture du réseau et leur connectivité.

Nous présenterons dans la section 3.2 les plates-formes à agents mobiles et la façon de les utiliser, puis dans la section 3.3 les itinéraires que l'on peut appliquer à des agents mobiles. Les études qui ont été faites pour déterminer la topologie des réseaux sont présentées dans la section 3.4 et nous conclurons ce chapitre dans la section 3.5 en donnant une orientation pour appliquer les agents mobiles dans un contexte d'administration système et réseau.

3.2 État de l'art des plates-formes à agents mobiles

A notre connaissance, il existe depuis le début des années 1990, c'est-à-dire depuis les premiers travaux du *Management by Delegation* [30] qui ouvraient la voie des plates-formes dans le cadre de l'administration système et réseau, un certain nombre de plates-formes à agents mobiles pour l'administration système et réseau, qui se révèlent être incontournables. Notre présentation décrit les principes de l'administration par délégation, puis nous présenterons les plates-formes à agents mobiles tournées vers l'administration système et réseau.

3.2.1 Principe de l'administration par delegation

L'approche introduite par MAD [30] présente la première les concepts de la délégation des tâches d'administration de la station centralisée vers des éléments intermédiaires, qui a leur tour deviennent des stations d'administration agissant comme des *proxies*. L'architecture de MAD, repose sur des possibilités de déporter les fonctions d'administration de la station centralisée vers ces proxy, appelés MAD agent.

Le composant central de l'architecture proposée par MAD est le *MAD agent* (voir figure 3.1). C'est une combinaison entre le modèle hiérarchique de supervision et d'un agent *proxy* (le MAD Agent) qui fournit un ensemble de services nécessaire pour effectuer la supervision, et le contrôle des éléments gérés. Le MAD

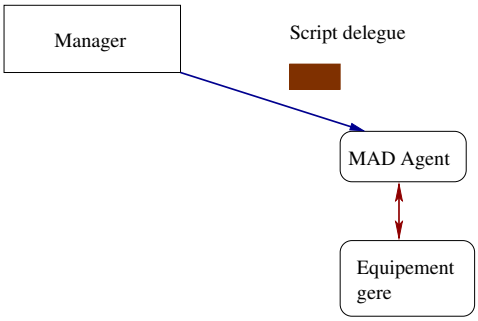

FIG. 3.1 – Architecture de MAD

Agent fournit un ensemble de services qui peuvent être invoqués depuis la station d'administration, ou depuis autre MAD Agent.

Le processus de délégation est engendré par la station d'administration sur laquelle ont été décrites les fonctions d'administration qui seront transmises à un MAD Agent. Ce processus permet de charger à la volée de nouvelles fonctions d'administration dans le *proxy* intermédiaire, ou de supprimer une fonction devenue inutile, tout cela pendant la durée de vie du MAD Agent. Dans l'architecture distribuée de MAD, un MAD Agent peut devenir à son tour le gestionnaire d'autres MAD Agents, et la station maître d'administration devenir un *agent proxy*. Tout cela grâce aux possibilités de l'architecture qui permet de rajouter ou de supprimer des fonctions dans ses nœuds.

Dans cette architecture, il existe trois types de fonctions possibles : la fonction autonome qui a son thread d'exécution et ses données privées, la fonction exécutée à la volée et qui retourne son résultat et la fonction qui est un objet à l'écoute des demandes de services, accessible par tous les composants de l'architecture. Le language de programmation de ces fonctions peut être le langage C ou tout autre language.

3.2.2 Plates-formes tournées vers l'administration

Suite aux travaux menés par MAD, un certain nombre de plates-formes à agents mobiles pour l'administration système et réseau, qui se révèlent être incontournables ont vu le jour et nous allons en détailler quelques unes, en présentant les points les plus originaux :

– MIAMI, projet européen basé au dessus de Grasshopper [50] : Grasshopper [94] est la première plate-forme à agents mobiles qui est compatible avec les standards industriels supportant des agents mobiles et leur gestion (OMG MASIF [14]). MIAMI regroupe plusieurs entités différentes, comme des entreprises, sous la forme d'un *Place-Oriented Virtual Private Network*, afin de pouvoir administrer des nœuds situés à différents endroits en appliquant, par exemple, la même politique de sécurité à tous ces nœuds. Cette approche permet de déléguer la gestion de ce *PO VPN* à une entité

extérieure à des fins d'administration.

- MAP [68] : Università di Catania, Italie.
  MAP sécurise en deux niveaux l'accès des agents mobiles sur les nœuds. Dans un premier temps lorsque les agents mobiles entrent dans un nouveau domaine (collection de nœuds regroupés sous la même autorité d'administration) ils obtiennent des droits d'accès sur les nœuds qui composent le domaine. De plus, une restriction peut être appliquée sur chaque nœud du domaine en fonction des exigences de sécurité mises en place par de l'administrateur.
- MAGENTA [76] : IRISA, Campus Universitaire de Beaulieu, France.
  La particularité de MAGENTA est de prendre en compte la déconnexion de la station d'administration. En effet, si l'administrateur utilise un ordinateur portable, il se peut qu'il soit déconnecté ou que le lien SLIP/PPP qui le relie au réseau soit très lent. Ainsi, lorsque les agents mobiles doivent revenir à la station d'administration, ils vérifient qu'ils ont la possibilité d'y revenir. Dans le cas contraire, l'agent mobile attend la reconnexion de la station avant d'y revenir. Pendant cette période d'attente, il stationnera sur un nœud de la plate-forme en attendant cet événement.
- SOMA [10] : Università di Bologna, Italie.
  Le point fort de SOMA est la supervision des agents mobiles et des nœuds d'accueil, et au passage des systèmes d'exploitation sous-jacents. Pour ce faire, on dispose d'une API permettant d'accéder, par exemple aux ressources suivantes : CPU, charge réseau, consommation de mémoire. En utilisant cette API, chaque nœud de la plate-forme peut être supervisé par un agent mobile dédié à cette tâche.
- James [84] : University of Coimbra, Portugal. Ce travail introduit la possibilité de superviser la plate-forme d'administration elle-même, c'est-à-dire James, en utilisant le protocole SNMP et l'instrumentation JMX pour tous les objets de la plate-forme (cf. annexe JMX 9.2). Pour cela la *MIB-JAMES* a été définie. Grâce à cette MIB, la plate-forme à agents mobiles peut aussi être supervisée par une plate-forme d'administration réseau classique (par exemple HP Network Node Manager).
- MobileSpaces [77] : National Institue of Informatics / Japan Science and Technology corporation, Japon. Cette plate-forme fournit des agents mobiles transporteurs d'agents mobiles. Ces agents mobiles transporteurs assurent les opérations de migration des agents qu'ils transportent dans un sous-réseau donné. Un ensemble d'itinéraires adaptés pour le sous-réseau est défini et mis à la disposition des agents mobiles transporteurs. La politique de migration, dans un sous-réseau donné, est dépendante des fonctions d'administration que doivent exécuter les agents mobiles. Ainsi, l'agent transporteur aura l'itinéraire associé à la fonction d'administration. Une extension possible de ces agents transporteurs est de les utiliser pour faire traverser un pare-feu aux agents mobiles d'administration, diminuant ainsi

les contraintes de sécurité qui sont mises en œuvre.

Nous avons listé des plates-formes à agents mobiles bâties uniquement au dessus du language Java. Mais il existe d'autres types de plates-formes à agents mobiles tournées vers l'administration système et réseau où les tâches à réaliser sont définies par d'autres languages, comme Tcl/Tk [92], qui permettent d'effectuer des opérations d'administration par l'utilisation de scripts, comme par exemple AgentTcl [31], AGNI [71], qui utilisent l'interpréteur de scripts pour distribuer le code dans les nœuds d'accueil et lancer des opérations d'administration à distance. Toutefois, un désavantage quant à l'utilisation des scripts repose sur la nécessité de l'interprétation du script, ce qui induit un temps d'exécution plus long, plutôt que d'utiliser un language compilé comme l'est Java.

Au lieu de faire un état de l'art qui décrirait chaque plate-forme d'administration à base d'agents mobiles l'une après l'autre, nous avons préféré organiser cette description en considérant tour à tour différent aspects d'importance :

- Installation de la plate-forme
- Programmation d'agents
- Communication entre agents
- Accès aux informations de la MIB SNMP

La notion d'itinéraire pour un agent mobile est si fondamentale que nous en avons fait l'objet d'une section à part (section 3.3).

3.2.3 Installation des plates-formes

Dans cette section nous aborderons le problème du déploiement des plates-formes à agents mobiles, qui se base sur la mise en route des nœuds et sur des techniques pour l'enregistrement de ces nœuds.

Lorsqu'il s'agit de travailler avec une plate-forme à agents mobiles, on introduit le concept de nœud. Un nœud dans une plate-forme à agents mobiles est une entité logicielle qui permet la réception, le départ et l'exécution d'un agent mobile. Ce nœud est une partie intégrante de la plate-forme. Pour cela le nœud fonctionne comme un `daemon` ou un `service`. Le nœud peut être mis en route systématiquement pendant la phase de démarrage du système (au boot). Le nœud peut être aussi démarré mais aussi ultérieurement arrêté à la demande de l'utilisateur, soit par des scripts automatiques de connexion, en utilisant des commandes à distance (par exemple `rsh`), ou encore en utilisant le plannificateur de tâches (ou le `crontab`) fourni dans le système d'exploitation.

Plutôt que d'utiliser des commandes systèmes pour démarrer ou arrêter les JVMs hébergeant les nœuds de la plate-forme, une solution intégrée à la plate-forme peut être plus adaptée. En effet, le problème de l'arrêt des nœuds à distance peut se révéler fastidieux s'il est fait manuellement et parfois problématique, car il faut en particulier identifier tous les processus système faisant tourner les JVMs.

Pour faciliter cela, il est possible de démarrer en tant que *daemon* du système une JVM, et de demander à cette JVM de créer une JVM en tant que processus fils qui elle hébergera un nœud de la plate-forme. L'intérêt c'est de pouvoir déléguer à ces JVMs daemon le problème du démarrage, de l'enregistrement et de l'arrêt des nœuds de la plate-forme. Cette technique, particulièrement intéressante dans le cas du redémarrage des nœuds est mise en œuvre dans James [81], la JVM *daemon* exécutant une classe s'appelant `Jrexec`.

Certaines contraintes dans le domaine de l'administration des systèmes et des réseaux sont importantes à considérer, même si elles n'ont pas fait l'objet de recherche au cours de cette thèse : il s'agit essentiellement de la sécurité. En effet, en fonction des droits de l'utilisateur démarrant le nœud, les agents mobiles ultérieurement hébergés par ce nœud auront plus ou moins de droits sur le système. Un nœud accessible à tout agent mobile de n'importe quel utilisateur, mais démarré avec les droits de l'administrateur (typiquement root) pose des problèmes de sécurité du système hôte. Néanmoins, vu que l'on veut faire de l'administration, il faut que les agents mobiles aient des droits correspondant à ceux de l'administrateur. Pour ce faire, une technique d'authentification de chaque agent mobile est souhaitable afin de les autoriser à effectuer des accès au système (commandes particulières sur le système).

Une fois les nœuds de la plate-forme à agents mobiles instanciés, il convient de trouver une méthode pour les localiser afin que ceux-ci soient accessibles aux utilisateurs. Selon les plates-formes, certains nœuds s'enregistrent automatiquement au moment de leur démarrage, comme dans le cas de MAP [69], en se connectant directement sur le nœud maître de la plate-forme. Cette solution a l'inconvénient d'introduire des références statiques entre les nœuds déployés. Une autre idée est de ne pas enregistrer les nœuds de la plate-forme mais de rendre ceux-ci accessibles par le biais de fichiers référençant les nœuds (méthode statique de mise à jour) comme c'est le cas dans James [82], ou en utilisant un bus à objet (par exemple `CORBA`) [9] pour l'enregistrement des nœuds en tant que service du réseau. C'est cette dernière idée qui offre le plus de souplesse car elle évite toute référence statique liée au déploiement de la plate-forme d'une part, et d'autre part elle laisse toute autonomie aux programmeurs pour enregistrer et localiser les nœuds de la plate-forme à agents mobiles.

3.2.4 Programmation des agents

3.2.4.1 Patterns de fonctions d'administration système et réseau à base d'agents mobiles

Plusieurs types d'agents mobiles peuvent exister. Ils sont à regrouper en fonction des tâches génériques qu'ils sont en mesure d'exécuter. Ils ont des possibilités de communication, entre eux ainsi qu'avec les autres éléments de la plate-forme,

de migration, de duplication, etc.. Nous détaillons ci-dessous les principaux types d'agents mobiles que nous avons rencontrés dans la littérature.

Statique : Un agent statique est un agent mobile qui n'exécute en général qu'une seule migration. Cette migration s'effectue depuis la station de départ vers un nœud bien défini au lancement. A l'arrivée sur le nœud, l'agent mobile ne migre plus mais exécute une tâche prédéfinie. Il peut s'agir d'un calcul de longue durée (par exemple vérifier périodiquement le nombre de requêtes exécutées sur une base de données), de la supervision de l'interface réseau (ce que fait par exemple le *Daemon Agent* de la plate-forme MAP [69]), de la supervision de l'espace disque du système hôte. Une autre utilisation d'un agent mobile statique serait de fournir une ou plusieurs listes d'éléments qui correspondraient à des itinéraires pour les agents mobiles de la plate-forme. Cette proposition est utilisée dans James [78] par le biais de l'*Agent Pool*. Un tel agent est aussi appelé un agent stationnaire car il ne migre plus pendant sa durée de vie.

Visiteur : Un agent visiteur est un agent mobile qui visite successivement les différents nœuds de la plate-forme afin d'y appliquer la même fonction d'administration. Il peut s'agir d'un agent mobile allant, par exemple, effacer sur chaque système les fichiers temporaires ou collecter les espaces disques utilisés, etc..

Le *Verifier Agent* [69] de MAP est un agent visiteur, dont l'utilisation peut faciliter les tâches de l'administrateur du réseau, par exemple, en récupérant les versions des systèmes d'exploitation installées sur les éléments du réseau.

Collecteur de données : Un agent collectant les données est un agent mobile qui nécessite un point de rendez-vous avec un ou plusieurs autres agents mobiles pour les décharger des données que ces derniers ont collectées. Cet agent de collecte peut toutefois récupérer des données qui ont été laissées à sa disposition sur les différents systèmes par d'autres agents mobiles. Après la collecte, cet agent transporte les données jusqu'à un système défini à l'avance.

Transporteur : Un agent transporteur est un agent qui transporte des agents mobiles d'un réseau à un autre, ou qui transporte les agents mobiles dans un sous-réseau. Il permet d'éviter la multiplication des procédures d'identification des agents mobiles lors du passage d'un réseau à un autre (par exemple si les contraintes de sécurité imposées aux agents mobiles diffèrent). Lorsque le transporteur agit uniquement dans un sous-réseau, celui-ci transporte un ou plusieurs agents mobiles qui ont les mêmes déplacement prévus (mêmes nœuds à visiter) (par exemple le *Navigator Agent* de MobileSpaces [78]).

Suivi d'incident : Un agent de suivi d'incident est un agent qui est programmé pour réagir aux données qu'il analyse. Il peut analyser les fichiers d'incidents ou

de fonctionnement d'un système (par exemple les `logs` Unix) et se déplacer, si nécessaire, sur un autre système (par exemple, détecté dans les logs) qui aurait été compromis.

Communication et synchronisation : Lorsque l'on veut utiliser un ensemble d'agents mobiles pour accomplir une tâche particulière (par exemple pour connaître l'état général d'un cluster), des techniques de communication et de synchronisation doivent souvent être mises en œuvre. En effet, les agents mobiles ont besoin de communiquer entre eux pour échanger leur état, leurs données, leur localisation si nécessaire afin de pouvoir mener à bien leur tâche. Plusieurs schémas existent :

- communication directe d'agent à agent (par exemple l'échange de la charge CPU d'un système)
- communication d'information d'un agent à une boîte postale (par exemple délestage des données transportées dans une base de données afin de pouvoir venir les récupérer ultérieurement)
- synchronisation entre agents (point de rendez-vous après l'exécution d'une tâche effectuée en parallèle)

3.2.4.2 Déploiement d'un agent

Nous avons décrit dans la section 3.2.3 de quelle manière sont lancés les nœuds d'une plate-forme à agents mobiles. Voyons maintenant quelle est la méthode "générique" qui permet de démarrer l'exécution d'un agent mobile sur de tels nœuds : `Agent monAgent = lanceAgent("Agent","nom du nœud")`.
Chaque plate-forme propose son propre son nom pour la méthode `lanceAgent`. Cependant, la mise en œuvre de cette méthode consiste toujours à instancier la classe "Agent" selon les modalités du noyau d'exécution.

3.2.4.3 Programmation d'un agent mobile

Pour faciliter la création d'agents mobiles dans une plate-forme donnée, on dispose en général d'un agent générique qu'il suffit de personnaliser puis d'instancier pour créer de nouveaux agents de ce nouveau type. Une classe (que l'on appelle Agent dans l'exemple de code figure 3.2) fournit le cadre général de - l'utilisation des fonctionnalités attendues d'un agent mobile dont notamment la possibilité de migration.

Dans une telle classe, on trouve en général une primitive (`go`) qui permet de faire migrer l'agent mobile de sa position actuelle vers un autre nœud de la plate-forme. Une primitive (dénommée par exemple `myWork` sur la figure 3.2) correspond à la fonction déclenchée automatiquement à l'arrivée sur un nouveau nœud. L'activité de l'agent mobile est matérialisée par une primitive (dénommée `run` sur la figure 3.2), qui est en fait celle exécutée par le thread qui "donne vie"

```
public class Agent{
  public void myWork() {    }
  public void go(String new_Node) {  }
  public void run() {
    // la vie de l'agent mobile
  }
}
```

FIG. 3.2 – Classe générique d'un agent mobile

et donc l'autonomie à l'agent mobile. L'utilisation de ce découpage en méthodes distinctes pour les opérations qui sont réalisées par un agent mobile est une approche utilisée dans Ajanta [3]. L'exemple de la figure 3.3 montre comment pourrait être personnalisé un agent générique.

```
public class HelloAgent extends Agent {
  public HelloAgent(Credentials credential) {
    super(credential);
  }
  public void run() {
    System.out.println("Agent"+credential.name+" says Hello to "+host.getHostURN());
    go("another_host_URN");
  }
  public void arrive() {
  }
  public void depart() {
  }
}
```

FIG. 3.3 – Classe HelloAgent de la plate-forme Ajanta

On y trouve une primitive `arrive` correspondant à la méthode à exécuter lors de l'arrivée sur le nœud, ainsi qu'une primitive `depart` qui est appelée avant le départ d'un nœud. L'appel de la méthode `go` est effectué dans la méthode `run` exécutée par le thread associé à l'agent mobile. Les itinéraires d'Ajanta sont des itinéraires typiquement construits manuellement et transmis à l'agent mobile au moment de sa création.

Un autre cadre de programmation est proposé par la plate-forme MAP [69]. Dans MAP, la programmation d'un nouvel agent mobile se fait par l'extension de la classe `Agent`. La méthode `exec` est la méthode qui est appelée après chaque opération de migration, méthode qui sera exécutée par le thread associé à l'agent mobile. Le programmeur doit implémenter sa fonction d'administration à partir du corps de la méthode `exec` (cf. figure 3.4). Dans cette méthode, l'appel de la méthode `go` permet de lancer le processus de migration de l'agent mobile. L'itinéraire qui est donné à l'agent mobile est contenu dans une variable globale, appelée `Locations`, qui contient la liste des nœuds à visiter et qui doit être mise à jour par retrait de la destination vers laquelle on va migrer pour éviter de boucler sur la même destination. Dans MAP, l'itinéraire est récupéré par la réception d'un

message émis depuis la station d'administration (méthode `receiveSyncMessage`) et entièrement suivi par l'agent mobile.

```
public class CirculateAgent extends Agent {
    Vector addressVector;

    public CirculateAgent()
    {
        addressVector = null;
    }
    // l'agent va de site en site tant que la liste adressVector contient des éléments
    public void exec()
    {
        while( addressVector == null ); // l'agent attend un itinéraire
        System.out.println( "************************************" );
        System.out.print( "         PARCOURS A EFFECTUER BIEN RECU:         " );
        Vector locations = loc.getAll();
        for( int i = 0; i < locations.size(); ++i )
        {
            System.out.println();
            System.out.print( ( ( URL ) locations.elementAt( i ) ).toString() );
        }
        System.out.println( " * mi trovo qui" ); // je suis là
        System.out.println( "************************************" );
        if( addressVector.size() != 0 ) //jusqu'à ce que ce soit fini
        {
            URL remoteURL = ( URL ) addressVector.remove( 0 );
            go( remoteURL );
        }
    }

    public Message receiveSyncMessage( Message m )
    {
        addressVector = ( Vector ) m.content;
        return m;
    }
}
```

FIG. 3.4 – Classe d'un agent mobile dans MAP

Suite à l'exposé de ces exemples (Ajanta et MAP), on peut constater que l'agent mobile gère lui-même son processus de migration, dans le sens où l'ordre de migration vient du code de l'agent mobile (c'est lui qui invoque la primitive `go`). Une approche différente et complémentaire, comme celle adoptée dans MobileSpaces [78] ou dans ProActive [8] est de pouvoir aussi déclencher la migration de l'extérieur de l'agent mobile. Cette possibilité permet, si besoin, de décharger les programmeurs du pilotage de la migration d'un agent. Ainsi, de se concentrer sur le développement du code de la tâche d'administration à réaliser.

3.2.4.4 Mécanisme de construction d'itinéraire

Fournir un itinéraire prêt à l'emploi pour un agent mobile pose le problème de la collecte des éléments qui font parties de l'itinéraire. Nous avons mis en évidence différentes méthodes pour obtenir des itinéraires qui sont utilisés par des agents mobiles.

Statique : Un itinéraire construit statiquement est un itinéraire qui est obtenu à partir d'un fichier décrivant un itinéraire. Ce type de construction est assez rigide et impose à l'administrateur de la plate-forme à agents mobiles de mettre à jour régulièrement cet itinéraire statique en fonction des nouveaux nœuds actifs de la plate-forme. Il s'agit ici de la solution la plus contraignante.

Endroit centralisé plus ou moins à jour : Il s'agit d'enregistrer automatiquement les nœuds de la plate-forme dans un nœud maître (Gavalas [63]). Cette technique permet de centraliser tous les éléments qui seront donnés pour les itinéraires en un seul lieu. Mais la défaillance d'un nœud n'est pas prise en compte parce qu'il n'y a pas de rétroactivité entre le nœud maître et les nœuds qui se sont enregistrés.

Au fur et à mesure du parcours : L'itinéraire obtenu est un itinéraire qui est construit à la volée. Il est découpé en portions et enrichi au fur et à mesure que l'agent mobile change de domaine. Ici le terme de domaine peut être un domaine logique regroupant certains nœuds de la plate-forme en fonction de besoins pouvant être très variés (par exemple, les machines d'une équipe, localisée sur plusieurs bâtiments), ou un domaine physique comme par exemple un sous-réseau. Ainsi à chaque changement de domaine, l'agent mobile devra enrichir son itinéraire avec les éléments à visiter sur le nouveau domaine. C'est une approche similaire qui est utilisée par MobileSpaces, car chaque agent mobile qui change de domaine est pris en charge par un agent mobile particulier qui pourra lui indiquer l'itinéraire à suivre dans le nouveau domaine.

3.2.4.5 Programmation de l'aspect fonctionnel

Conformément au schéma général d'un agent mobile (voir section 3.2.4.3), on s'intéresse à présent plus précisément à la programmation de la tâche d'administration qu'un agent devra exécuter en lieu et place d'un administrateur. Précédemment, nous avons vu que la primitive `go` définit la méthode (par exemple `arrive`) qui sera exécutée à l'arrivée sur le nouveau nœud. Pour permettre l'exécution de tâches d'administration, il est nécessaire de programmer cette fonction avec la tâche d'administration que l'on souhaite voir appliquée sur le nœud ou à partir de ce nœud. Il existe deux types de tâches d'administration que l'on veut voir réalisées par des agents mobiles : les tâches d'administration des systèmes (par exemple avoir la charge CPU) ; les tâches d'administration réseau (utilisation d'un protocole d'administration de réseau, SNMP en l'occurrence).

Fonctions d'administration de système : Les fonctions de ce type sont des fonctions qui font des appels systèmes afin de collecter, modifier ou installer des données ou des programmes. Dans un cadre de développement en Java, ces fonctions feront appel à l'interface native de Java, le `JNI`.

Par exemple, en utilisant le `JNI`, pour connaître la charge CPU d'un système quelconque, il faudra déterminer le type de système afin d'adapter le code du programme. En effet, la charge CPU est obtenue de manière différente en fonction des systèmes d'exploitation. Sous Linux, en lisant le fichier `/proc/loadavg` tandis que sous les environnements Windows, il faudra développer la bibliothèque (une DLL, dynamic link library) fournissant cette information.

Fonctions d'administration réseau : Pour écrire des fonctions d'administration réseau, il faudra utiliser le protocole SNMP, car c'est le protocole d'administration de réseau le plus répandu. L'implémentation du protocole SNMP existe, par exemple proposée par AdventNet [2] dans sa bibliothèque Java, afin de faciliter l'écriture de ces fonctions. Une fois que le protocole est disponible, n'importe quelle fonction d'administration réseau peut être écrite, comme par exemple obtenir le nombre et le type des interfaces réseau d'un système.

A titre de récapitulatif, nous donnons le cadre de programmation de SOMA [9] (cf. figure 3.5) : la primitive `run` pour définir l'activité principale de l'agent mobile, `goNode` pour déclencher le processus de migration et la primitive `verifyNode` qui est la fonction d'administration système à exécuter sur le nœud d'arrivée. Dans cet exemple, la fonction `verifyNode` permet de récupérer l'état du système au passage de l'agent mobile. En suivant son itinéraire, l'agent mobile pourra ainsi collecter l'état de tous les systèmes visités. On remarque un aspect intéressant qui est le fait de pouvoir associer le nom du nœud à visiter et le nom d'une méthode qui doit être exécutée à l'arrivée sur ce nœud. Ce nom de méthode est ici quelconque, contrairement à ce que l'on avait pu observer par exemple dans le code de la figure 3.3 (méthode de nom `arrive`) ou dans celui de la figure 3.4 (méthode de nom `Exec`).

3.2.5 Communication entre agents mobiles

Il est essentiel que les agents mobiles puissent avoir un moyen de communiquer entre eux ou avec l'extérieur, sans avoir à se préoccuper de leur localisation. Les plates-formes évoquées au début de ce chapitre, sont reprises ici en fonction des moyens de communication qui sont fournis aux agents mobiles.

- MIAMI : Une définition d'une boite aux lettres sert de moyen de réception des communications entre les agents mobiles s'envoyant des messages ayant pour adresse : nom@domaine. Domaine est le domaine où se situe actuellement l'agent destinataire du message. L'agent mobile ne changeant pas forcément de domaine lors de la migration. Les agents mobiles peuvent tout de même se localiser entre eux, puisque les facilités requises par MASIF [14] en terme de localisation des agents mobiles (service d'enregistrement utilisant le Bus Corba, RMI, etc..), sont implémentées en standard dans la plate-forme Grasshopper.

```
void run() { // starting method for every Agent
  // Asking to AgentSystem the list of Nodes in this Domain
  Node = AgentSystem.getAllDomain();
  CurrNode=0;

  // Looking for the first active Node
  for (;CurrNode<Node.length;CurrNode++)
    if (AgentSystem.isActive(Node.Name)) goNode();
      .. // Error: no active nodes
}

void goNode() {
  try {
    this.go(Node[CurrNode].Name,"VerifyNode");
  }
  catch(Exception e) { //Can't go, System or Security exception
    ... goHome(); // Back home with failure status
  }
}

void verifyNode() { // Restart method specified by goNode()
  try {
    CPUload[CurrNode]=Monitor.getCPULoad();
  }
  catch(Exception e) { // action not allowed
    ... // Actions for exception handling
  }

  CurrNode++;
  for (;CurrNode<Node.length;CurrNode++)
    if (AgentSystem.isActive(Node.Name)) goNode();
  goHome();
}
```

FIG. 3.5 – Classe d'un agent mobile dans SOMA

- MAP : les messages sont échangés en connaissant le lieu et le port TCP du MAP Server (nœud de la plate-forme), et si l'agent n'est pas sur le nœud, alors le message n'est pas réacheminé. Le système utilise une liste des MAP Servers pour essayer de localiser l'agent. En cas d'échec, le message ne peut pas être retransmis.
- MAGENTA : Lorsque deux agents ont besoin de communiquer, ils se fixent un rendez-vous sur le même site, et s'échangent des messages pour communiquer. Lorsque la station d'administration veut communiquer avec les agents mobiles elle utilise un service de localisation inclues dans la plate-forme.
- SOMA : Dans le cadre de cette plate-forme, les agents mobiles s'échangent des messages pour communiquer entre eux. Pour se localiser dans le domaine de SOMA, un agent mobile utilise les facilités offertes par *MAFFinder* [14], une sorte de service d'enregistrement que les agents mobiles interrogent afin de pouvoir se localiser. Chacun des nœuds de la plate-forme enregistrant au fur et à mesure les agents mobiles qu'ils hébergent.
- James : Inter-agent communication avec le JavaSpaces, où tous les agents

s'enregistrent afin de pouvoir communiquer entre eux. Une autre méthode pour fournir des communications entre les agents mobiles et d'utiliser le nœud maître de la plate-forme qui possède la liste de tous les agents mobiles en activité dans la plate-forme. Par ce biais, un agent mobile localise indirectement l'agent mobile avec lequel il souhaite communiquer.

– MobileSpace : Chaque agent maintient une file de requêtes de messages et il s'enregistre auprès des autres agents avec lesquels il doit converser (donc défini statiquement). Lorsqu'un agent doit bouger, il laisse derriere lui un agent forwarder, qui permet à tous autres agents mobiles visitant le nœud, de savoir le recontacter par le bias des forwarders. Ces agents mobiles utiliseront le *forwarder agent* pour migrer à la même localisation que l'agent mobile avec lequel ils doivent communiquer, et pourront ensuite dialoguer avec lui.

Les mécanismes de localisation et de communication entre les agents mobiles doivent être transparents aux agents mobiles et c'est le rôle de la plate-forme de fournir de tels mécanismes. Comme nous verrons par la suite, la plate-forme ProActive a le mérite de fournir ces mécanismes de manière que l'on peut qualifier d'idéale !

3.2.6 Accès aux données de la MIB SNMP

Afin de pouvoir accéder aux informations de la MIB SNMP, il est utile que les agents mobiles puissent converser avec les agents SNMP.

Certaines plates-formes ont intégré dans leur distribution des classes permettant de converser en SNMP. Cela implique que lorsqu'un agent mobile arrive sur un nœud sur lequel il doit exécuter une fonction d'administration requérant le protocole SNMP, il n'a pas besoin de télécharger les classes. Ces classes mettent en œuvre une API qui, pour masquer la complexité de l'utilisation du protocole SNMP, prédéfinit un ensemble de fonctions d'administration (par exemple, l'obtention de statistiques sur des interfaces réseaux). L'inconvénient évident est que certaines requêtes SNMP ne pourront pas être réalisées. Cette solution est mise en œuvre, par exemple par SOMA [10], qui intègre une interface (nommée *Monitoring Application Programming Interface*), permettant de fournir une agrégation des résultats des indicateurs de surveillance. Ainsi l'agent mobile arrive et collecte directement la ou les valeurs des indicateurs.

Une autre méthode est celle qui consiste à associer le code de l'agent mobile avec une API indépendante de la plate-forme qui donne accès à toute la pile du protocole SNMP (méthodes, fonctions, encodages, décodages, etc..), comme par exemple `AdventNet` [2]. En utilisant cette technique, l'agent mobile est libre d'accéder à n'importe quelle information contenue dans la MIB de l'agent SNMP. L'agent mobile va devoir télécharger les classes utiles pour réaliser sa fonction d'administration SNMP, ce qui peut être pénalisant en terme de performance si le lien réseau entre l'hôte de départ et l'hôte d'arrivée est de faible débit.

3.3 Itinéraires

3.3.1 Motivation

Pour caractériser le concept d'itinéraire [4, 89], nous dirons que l'itinéraire d'un agent mobile est l'union d'opérations composées d'un déplacement (communément appelée une migration) et d'une action que l'agent mobile doit exécuter à chaque point de passage de cet itinéraire. Un itinéraire est un concept important qui permet de décharger l'agent mobile de la gestion explicite des opérations de migration, et lui permettre de se concentrer uniquement sur son objectif principal : exécuter la ou les tâches en lieu et place d'un administrateur.

3.3.2 Structuration d'un itinéraire de visite

Comme on vient de le voir (cf. section 3.2.4.4), les éléments d'un itinéraire peuvent être collectés de manière statique par l'administrateur de la plate-forme à agents mobiles (liste d'éléments pré-définis pour un itinéraire), de manière automatique (enregistrement automatique des nœuds au démarrage de la plate-forme), ou par un système autonome qui collectera les éléments à indiquer dans l'itinéraire.

Quelque soit le moyen utilisé pour mettre ces éléments à disposition, on peut remarquer qu'un itinéraire peut être structuré de différentes manières : plate ou par domaine (virtuel ou physique) à visiter.

3.3.2.1 Structure plate

C'est la structure la plus simple d'un itinéraire couvrant tous les éléments au sein du réseau d'entreprise, par exemple, et ce sans aucune organisation quelconque liée à la localisation de ces éléments.

3.3.2.2 Structuration virtuelle

C'est une structuration possible d'un itinéraire dans lequel les éléments sont regroupés virtuellement (dans ce que l'on peut appeler un domaine). On entend par virtuel, le fait que des éléments regroupés n'appartiennent pas tous au même réseau, ou au même sous-réseau, mais dont les fonctionnalités permettent de les regrouper dans une entité administrable logiquement. Par exemple, un itinéraire couvrant certains éléments pris dans chaque sous-réseau d'une entreprise pourra donner lieu à un itinéraire structuré virtuellement et donc définissant un domaine dans le réseau. Il sera donc possible d'avoir plusieurs itinéraires de ce type sachant qu'il est possible de créer plusieurs domaines virtuels d'éléments dans un réseau.

SOMA [9] propose une telle solution (cf. figure 3.6) dans le but de gérer les droits et les déplacements des agents mobiles au sein de la plate-forme, grâce à des regroupements virtuels d'éléments. Chaque nœud de la plate-forme est créé

avec comme référence le nœud maître du domaine dans lequel il fait partie. Un domaine peut être défini comme étant un sous-réseau entier, mais aussi englober des nœuds appartenant à un autre sous-réseau. Dans cette approche, les domaines sont sous la responsabilité d'un ou plusieurs administrateurs. Le déplacement des agents mobiles d'un domaine à l'autre se fait obligatoirement en passant par une entité de contrôle (nœud maître) qui alloue automatiquement les nouveaux droits aux agents mobiles requis pour accéder au domaine visé.

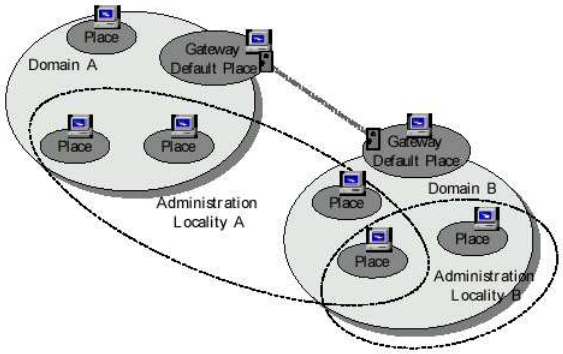

FIG. 3.6 – Modèle des domaines de SOMA

3.3.2.3 Structuration physique

C'est la mise à disposition d'un itinéraire couvrant les éléments connus appartenant au même sous-réseau. Il existera donc plusieurs itinéraires de ce type dans le réseau d'une entreprise, par exemple, si celui-ci est composé de plusieurs sous-réseaux.

Une approche par sous-réseau est une approche qui colle à la réalité de l'administration. Lorsqu'un agent mobile doit changer de réseau, il contactera le service d'itinéraire présent sur le réseau cible. Il existe deux variantes de cette solution : la première masque les itinéraires aux agents mobiles ce qui introduit la présence d'un agent mobile de "transport" (voir section 3.2.4.1) pour fonctionner ; la deuxième fournit directement aux agents mobiles des itinéraires ce qui donne la possibilité aux agents mobiles de gérer eux-mêmes le déroulement de leur itinéraire, voire de le modifier si nécessaire.

L'approche par agent mobile transporteur est présente dans MobileSpaces [78] sous la forme d'un *Agent Pool* et d'un *Navigator Agent* (cf. figure 3.7). L'*Agent Pool* est un système qui est initialisé avec tous les itinéraires du sous-réseau et qui les modifie en fonction des conditions rencontrées par les agents mobiles (par exemple, mise à jour en cas de nœud défaillant). Chaque *Navigator Agent* possède un itinéraire pré-déterminé par l'*Agent Pool* et il est à disposition des *Task Agent* (agents mobiles exécutant des fonctions d'administration), afin de les transporter

sur le sous-réseau. Le *Navigator Agent* transporte un ou plusieurs *Task Agent*, il gère tout le processus de la migration, de l'exécution des ordres de migration. Le *Navigator Agent* met à jour à son retour la liste des éléments qui sont devenus inaccessibles sur le sous-réseau, détenue au sein de l'*Agent Pool*.

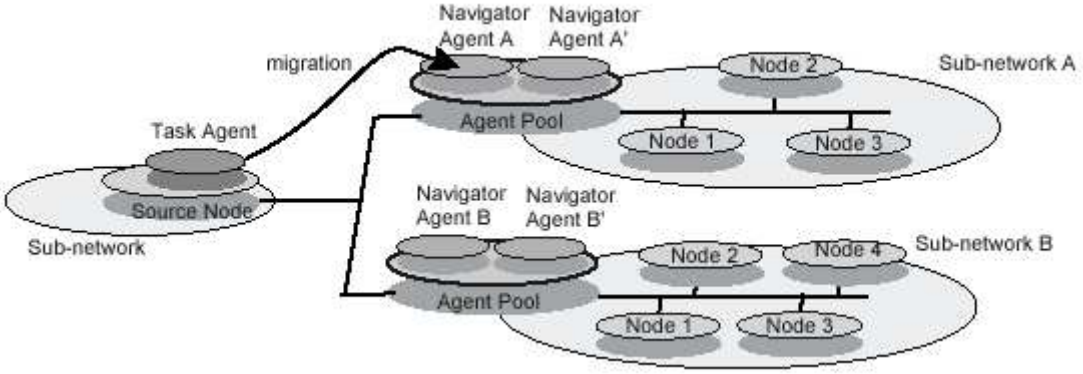

FIG. 3.7 – Les itinéraires dans MobileSpaces

3.3.2.4 Composition

Pour donner plus de souplesse aux itinéraires, une solution de composition de ces éléments peut apporter un avantage pour l'autonomie et la réactivité des agents mobiles. En effet, il est possible d'imaginer des itinéraires globaux (section 3.3.2.1) ; des itinéraires structurés par domaine (section 3.3.2.2) ; des itinéraires structurés par sous-réseaux (section 3.3.2.3). Avec une telle composition, il sera possible de fournir des itinéraires couvrant tout le domaine d'administration, un sous-réseau ou un ensemble d'éléments répartis sur plusieurs sous-réseaux de l'entreprise.

3.3.3 Autres structurations d'un itinéraire

Jusqu'à présent, nous avons décrit des itinéraires qui sont des itinéraires non structurés dans le sens où il s'agit de parcourir séquentiellement les éléments les uns après les autres. On constate toutefois que nous avons besoin d'itinéraires plus sophistiqués plutôt qu'un simple parcours. Il faut pouvoir donner des schémas de migration afin d'adapter le comportement des agents mobiles en fonction des conditions qu'ils rencontrent pendant leur phase de migration. Les itinéraires sophistiqués sont des itinéraires qui peuvent prendre en compte les événements qui surviennent dans le fonctionnement général d'un réseau : coupure d'électricité, panne réseau, lien défaillant, panne système, redémarrage d'un élément, surcharge du CPU système, surcharge de l'interface réseau, en sont des exemples concrets. Il faut aussi prendre en compte les tâches qui peuvent être effectuées en parallèle

sur différents systèmes. Ce parallélisme introduit des notions de synchronisation entre les différents composants.

3.3.3.1 Sélection de la Destination

La sélection de la destination parmi un ensemble de destinations peut être vue comme un itinéraire particulier intégrant une notion de *sélection*. Cette sélection ou ce choix entre les différents éléments à visiter est basée sur des critères de charge CPU des éléments, de la charge du réseau, par exemple. Autre exemple, la visite d'un élément d'un cluster de PCs peut suffire à obtenir l'information recherchée, sans avoir à visiter chacun des éléments du cluster.

3.3.3.2 Visite au hasard

Un tel type d'itinéraire consiste en la visite au hasard de l'ensemble des éléments appartenant à ce type d'itinéraire. Il n'y a pas de priorité assignée à chaque élément de l'ensemble, mais chaque élément doit être visité. Le critère de choix de la destination dans cette visite au hasard, peut être basé, par exemple, sur le plus court chemin à parcourir pour y arriver, la puissance de l'hôte, etc... Il s'agit, par exemple, de surveiller l'espace disque disponible sur un ensemble d'éléments du réseau sans qu'il y ait un caractère d'urgence. Ici, l'ordre de visite n'apporte rien de plus puisque seul le résultat compte.

3.3.3.3 Parallélisme

Il s'agit d'utiliser des notions de parallélisme pour faire exécuter la même tâche à des agents mobiles en parallèle. Ce mode de fonctionnement peut imposer des types de synchronisation pour la collecte des informations en retour :

- attente d'au moins un agent mobile pour continuer le parcours de l'itinéraire principal
- attente de tous les agents mobiles pour continuer le parcours de l'itinéraire principal

On peut, par exemple, utiliser cette notion de parallélisme dans le cas d'un réseau sans fil (Wifi). Nous savons pertinemment que la bande passante d'un réseau Wifi est faible par rapport aux réseaux câblés. Pour effectuer une tâche d'administration nécessitant un délai d'exécution court, un itinéraire parallèle est approprié dans ce cas là. En effet, la même tâche pourra être faite en parallèle sur chacun des ordinateurs connectés au réseau sans fil, et le temps d'exécution de la fonction globale sur l'ensemble des ordinateurs sera inférieur à une fonction exécutée séquentiellement sur chacun des ordinateurs les uns après les autres. De plus, on peut espérer profiter d'un recouvrement calcul/communication : pendant qu'une des tâches a déjà commencé à s'exécuter, les agents mobiles peuvent continuer à transiter sur le réseau sans fil pour atteindre le nœud de destination.

3.3.3.4 Un exemple de mise en œuvre de la composition de destinations dans un itinéraire

Le mécanisme d'itinéraires mis en œuvre dans Ajanta [3] consiste en une composition sophistiquée d'actions et de migrations. Chaque élément de l'itinéraire est constitué par ce qui est appelé un *pattern* de migration. Un itinéraire est une séquence de tels *patterns*. Chaque *pattern* est associé avec une action que l'agent mobile devra effectuer lorsque l'agent mobile arrivera sur un nouvel hôte, dont le descriptif est dans l'itinéraire (par exemple, appeler la fonction de nom *justOnTime*).

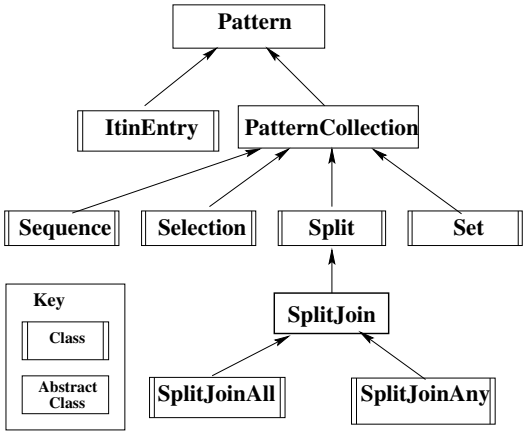

FIG. 3.8 – Les *Patterns* de migration dans Ajanta

Les *patterns* de migration de Ajanta sont les suivants (cf. figure 3.8) :
- Pattern : Élement abstrait de base d'un itinéraire
- ItinEntry : La définition d'un hôte à visiter et l'appel de la fonction
- PatternCollection : Une agrégation de *pattern* de migration
- Sequence : L'agent mobile visite la liste des hôtes de façon séquentielle
- Selection : L'agent mobile visite un hôte des hôtes de la liste des serveurs
- Set : L'agent mobile visite la liste des hôtes au hasard
- Split : L'agent mobile peut créer des agents mobiles fils qui feront le travail à sa place, en parallèle
- SplitJoin : L'agent mobile père attend le retour de tous les agents mobiles fils et il se synchronise avec leur résultat
- SplitJoinAll : L'agent mobile père attend tous ses agents mobiles fils dans un état de rendez-vous
- SplitJoinAny : L'agent mobile père attend le retour d'un seul de ses agents mobiles fils

En effectuant une combinaison des éléments du schéma proposé par Ajanta, on peut obtenir un itinéraire d'administration évolué. Les possibilités offertes par le parallélisme et les points de rendez-vous, permettent de réaliser des tâches à grande échelle avec le même type d'agent mobile.

3.4 La topologie du réseau

3.4.1 Motivation : besoin de découverte automatisée

Il s'agit d'étudier les outils qui permettent de découvrir la topologie d'un réseau, que ce soit au sein d'un réseau local, ou dans un réseau plus étendu comme un réseau d'entreprise, métropolitain ou longue distance.

Breitbart et al. [13] remarquent qu'il est difficile de conserver la topologie du réseau manuellement et que ce travail devient vite frustrant. En effet, un ajout de routeur ou d'un commutateur entraîne un changement dans le fonctionnement global du réseau et une cartographie automatique permet de faire aisément gagner du temps à l'administrateur. Le travail de Schonwalder [79] propose de déterminer la topologie du réseau au niveau 3 parce que le protocole IP est le protocole le plus répandu. C'est d'autant plus facile à réaliser parce qu'il existe des outils dans la pile du protocole IP (par exemple le drapeau *ICMP* dans un paquet IP). Ainsi, il est aisé de déterminer la topologie d'un réseau de niveau 3, car tous les équipements actifs intervenant savent naturellement dialoguer en utilisant le protocole IP. On trouve dans la littérature des algorithmes permettant la construction de la topologie de niveau 3 et récemment grâce à l'amélioration des équipements actifs (commutateurs, concentrateurs intelligents), on trouve des algorithmes pour construire la topologie de niveau 2 pour les réseaux locaux.

3.4.1.1 Définition de la notion de topologie

La topologie de niveau 3 est la cartographie d'un réseau qui répond au protocole IP. Les équipements actifs intervenant dans cette topologie sont des équipements capables d'effectuer du routage de paquets IP (par exemple un routeur). Les autres équipements sur le chemin, tels que les commutateurs et les concentrateurs ne peuvent pas être répertoriés puisqu'ils ne participent pas à la gestion du trafic IP.

La topologie de niveau 2 est la cartographie d'un réseau local (LAN) prenant en compte tous les équipements actifs intervenant dans le fonctionnement et la transmission d'information. On rappelle qu'un équipement actif de niveau 2 ne participe pas au cheminement de paquets IP dans le réseau (pas de routage, par exemple). Il n'est pas trivial de détecter simplement des équipements actifs de niveau 2. Un équipement actif de niveau 2 peut être détecté, s'il possède un agent SNMP actif, lors de la construction de la topologie. Les services que l'équipement actif fournit, sont définis dans le descriptif du système de la MIB-2 SNMP (*mib-2.system.sysServices*). Par contre, un équipement simple, tel un concentrateur ou un commutateur sans agent SNMP, ne pourra pas être détecté. C'est en posant quelques hypothèses et en faisant quelques regroupements d'information, que l'on espère justes, que l'on pourra les intégrer dans la topologie

construite (par exemple, si plus d'une adresse Ethernet est détectée sur un port d'un commutateur, alors on peut considérer qu'il y a un concentrateur connecté sur ce port du commutateur).

En synthétisant les différentes définitions de ce qu'est la topologie, on peut dire que la topologie du réseau est une vue de l'ensemble des éléments du réseau (LAN, MAN, WAN) qui entrent en jeu dans le fonctionnement des services qui sont fournis par le biais du réseau. Par le biais de cette topologie, les administrateurs réseau peuvent avoir une vue globale des équipements actifs qui fonctionnent (les équipements actifs défaillants seront détectés par des alarmes provenant de ceux opérationnels) et qui permettent de fournir un service : l'accès au Web, l'accès aux services de messagerie, en sont des exemples.

3.4.1.2 Exemples d'applications liées à la connaissance de la topologie

Pour compléter la motivation de disposer de la cartographie du réseau, nous donnons ci-dessous quelques exemples concrets où cette topologie s'avère intéressante.

Surveillance des liens L'apparition des applications réparties pose de plus en plus de problèmes aux administrateurs, tant par les besoins de qualité de services induits que par la bande passante réseau qui est utilisée par ces applications. La localisation des serveurs d'application peut être répartie sur tout le réseau de l'entreprise ce qui peut induire des surcharges locales sur certains brins du réseau. La bonne connaissance de la topologie permet à l'administrateur de redimensionner le débit de son réseau ou de délocaliser certains serveurs d'applications en fonction de la plus forte demande.

PathFinder [33] propose de connaître la topologie dans le but de savoir comment circulent les flux sur le réseau, de quel poste client vers quel poste serveur, etc.., sachant que tout poste client joue de plus en plus souvent aussi le rôle d'un serveur d'information ou de calcul du réseau. Les possibilités de partage de données ou la puissance de calcul des postes de travail actuels permettent en effet à tout poste "Client" de devenir un poste "Serveur" sur le réseau.

Filtre pour les alarmes du réseau Pour améliorer la qualité des alarmes qui remontent du réseau, une bonne connaissance de la topologie du réseau peut s'avérer nécessaire. Chaque élément du réseau peut renvoyer une alarme après avoir détecté une faute du réseau, ce qui face à l'augmentation croissante des éléments (commutateurs, PCs, etc..) peut engendrer beaucoup d'alarmes. L'utilisation de la connaissance de cette topologie permettrait de filtrer les alarmes pour n'obtenir que quelques alarmes permettant à l'administrateur de prendre une décision rapide.

Ainsi Breitbart [13] propose de définir un algorithme pour construire une topologie de réseau physique de niveau 2 permettant de filtrer les alarmes (sup-

pression des alarmes redondantes provenant de différents éléments ayant détectés une panne). Avec cette technique, l'administrateur pourra focaliser son attention sur l'élément du réseau (en général un équipement actif) qui est devenu défaillant.

Observation des performances du réseau Lorsqu'un réseau Ethernet est segmenté en utilisant des commutateurs, chaque commutateur obtient une vision du segment qu'il gère. Sur chaque segment, l'administrateur peut collecter sa charge en utilisant les informations du commutateur. Avec la corrélation des vues de tous les équipements actifs, l'administrateur peut détecter les endroits de son réseau consommant le plus de bande passante, voire même saturant les segments. Par cette correlation, l'administrateur peut intervenir afin d'améliorer les conditions générales du réseau, soit en déplaçant physiquement les serveurs d'application, soit en gérant mieux le trafic de chaque segment (utilisation de seuil par exemple pour limiter la bande passante de certaines applications).

Une approche proposée par REMOS [46] et par Fontanini [28], est d'utiliser la topologie du réseau afin d'essayer d'anticiper des problèmes de charge sur le réseau pendant des phases critiques mais aussi de pouvoir analyser le réseau en mode de fonctionnement normal.

3.4.2 Techniques de découverte de la topologie

Dans cette section nous présenterons les techniques utilisées pour déterminer la topologie des réseaux de niveau 3 (IP) et dans la section suivante celle de niveau 2 (Ethernet). Nous commencerons notre présentation par les outils commerciaux qui utilisent ces techniques et nous montrerons quelles ont été les recherches effectuées sur les deux niveaux par la suite.

3.4.2.1 Les outils disponibles dans les plates-formes d'administration commerciales

De plus en plus d'outils commerciaux se sont penchés sur l'obtention de la topologie du réseau. Les premières topologies de réseaux construites ont été faites au niveau de la couche réseau (niveau IP). Les principaux outils qui relèvent de ce type de construction de la topologie du réseau sont évidemment Hp Network Node Manager [35] et IBM Tivoli [95]. Ces deux outils commerciaux ont utilisé les informations qu'il est possible de trouver dans les MIBs SNMP des routeurs pour construire leur topologie. Avec la diminution des coûts de la bande passante et l'amélioration du cœur du réseau (matériel actif plus performant comme par exemple des commutateurs), on trouve désormais des outils commerciaux pour avoir une topologie de niveau liaison de données (niveau Ethernet, ATM, etc..). Le principal outil commercial réalisant une telle tâche est l'outil Hp Network Node Manager extended Topology [35] qui fournit via le même outil une topologie de niveau 3 et une topologie de niveau 2. Dans les sections suivantes nous parlerons

des outils de construction de la topologie dont l'algorithme est disponible, parce que dans les outils commerciaux il n'est pas possible d'avoir la ou les méthodes permettant de construire la topologie du réseau.

3.4.2.2 Topologie de Niveau 3

La topologie de niveau 3 permet de connaître la connectivité qui peut exister entre les différents éléments du réseau de l'entreprise, même d'une portion d'Internet, et ce en utilisant les techniques traditionnelles disponibles dans les systèmes d'exploitation actuels. Les principaux outils ont été énumérés dans le chapitre 2. Il s'agit des outils de base comme *Ping*, *Arp* et *Traceroute* (voir la section 2.3.3). Les équipements actifs composant le cœur du réseau n'interviennent pas dans le routage du trafic et ne peuvent pas être mentionnés dans une topologie de Niveau 3.

Les premières recherches pour déterminer une topologie de niveau 3 ont été effectuées par Siamwalla [70]. La technique utilisée permet d'avoir la topologie d'un WAN, d'un MAN ou tout simplement d'un LAN en ne prenant en compte que les routeurs.

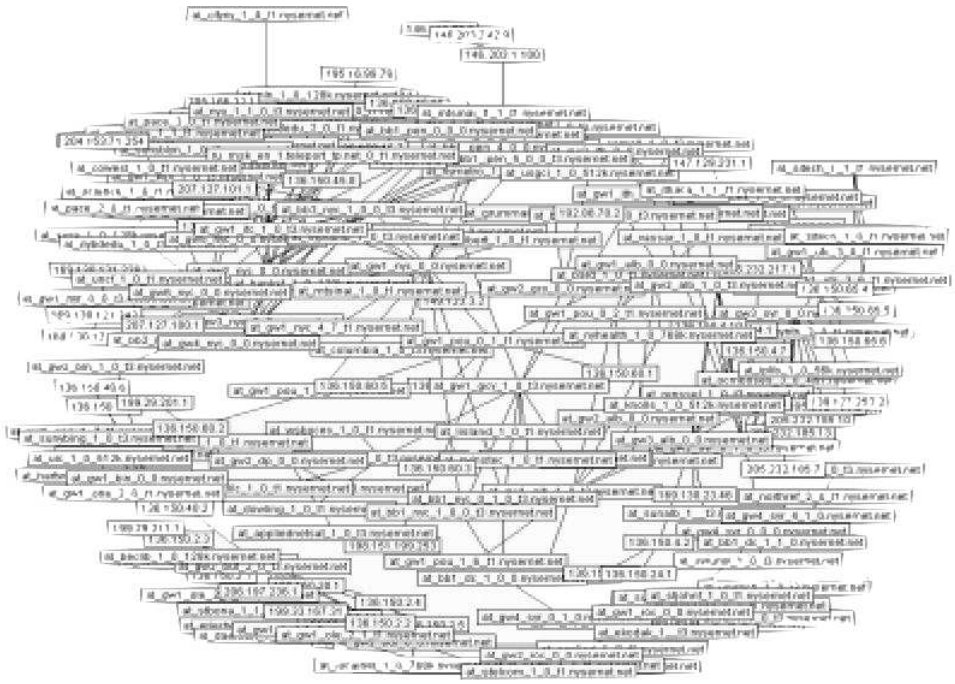

Fig. 3.9 – Une portion de l'Internet

L'algorithme proposé par Siamwalla [70] (voir topologie résultante figure 3.9) est une combinaison des techniques vues dans la section 2.3.3. Il s'agit d'utiliser les informations du protocole IP (Ping et Traceroute), de rajouter la possibilité

d'interrogation des serveurs DNS [26] (DNS Zone Transfert : permet d'obtenir la liste de toutes les machines enregistrées dans le serveur DNS) pour construire la topologie désirée. Celle-ci est effectivement trouvée avec plus ou moins de rapidité (dépend du nombre d'hôtes dans la classe d'adresse IP) car l'outil doit tout d'abord déterminer le masque du réseau associé à chaque sous-réseau qui doit être analysé. En effet, la recherche des adresses IP est résolue en diminuant le nombre de bits du masque du réseau (de 31 à 7, c'est-à-dire couvrant les classes d'adresse IP C, B et A), puis en envoyant des paquets *ICMP* en broadcast sur le réseau. Les éléments qui répondent permettent de déterminer le masque de réseau associé aux adresses IP. Chaque élément qui répond est répertorié en utilisant les principes du système de fichier Unix, c'est-à-dire la classification arborescente des répertoires et des fichiers. Chaque élément est enregistré dans le répertoire dont le nom est défini par l'adresse IP du réseau. Ainsi avec cette technique, Siamwalla [70] obtient la représentation d'une topologie sous forme arborescente. Seuls les éléments de types *routeurs* sont présentés sur le graphique (cf. figure 3.9) de la topologie obtenue.

Les techniques utilisées dans la proposition de Schonwalder [79] sont d'utiliser la combinaison du protocole ICMP pour déterminer la connectivité des éléments appartenant à la topologie finale et les outils comme *traceroute* pour déterminer les routeurs intervenant entre les réseaux ciblés. Une fois ceux-ci détectés, l'adresse des réseaux est obtenue en utilisant le *ICMP mask request*. Le protocole ICMP permet de demander au routeur gérant le sous-réseau de retourner l'adresse du réseau et le masque d'adresse associé. L'utilisation du masque de réseau permet de regrouper les éléments découverts dans le même sous-réseau dessiné graphiquement.

Une extension des techniques de Schonwalder [79] est proposée par Lin [34]. Lorsque les éléments ont été détectés, Lin [34] utilise les informations des agents SNMP.

Ainsi, ceci permet de borner simplement l'étendue de la découverte de topologie de niveau 3. En effet, la découverte se fonde sur la possibilité d'interroger les agents SNMP des équipements actifs. Dès lors que l'on n'a pas accès aux informations des MIB-2 SNMP (car elles seraient sous la responsabilité d'un autre administrateur de réseau), la découverte s'arrête d'elle-même. En plus cette méthode permet d'améliorer sensiblement le temps de construction de la topologie par rapport à [79], parce qu'il est possible d'avoir puis d'exploiter les informations des tables de routage.

Une approche différente est d'essayer d'utiliser une plates-formes à agents mobiles pour construire la topologie du réseau IP. Par exemple, un agent capable de se déplacer sur le réseau distant effectue la collecte des adresses IP du dit réseau, revient à la station d'administration principale et repart, si nécessaire vers un nouveau sous-réseau à découvrir. Une telle approche, mettant en évidence l'intérêt de la mobilité est présentée par la plate-forme AGNI [71], montrant que l'effect de délocalisation permet d'améliorer de près de 30 % le temps global de

la construction de la topologie en utilisant les agents de AGNI, par rapport à la solution centralisée.

3.4.3 Topologie de niveau 2

3.4.3.1 Introduction

Avec la somme d'information disponible de fait dans certains équipements actifs du réseau (niveau 2), il devient possible de découvrir la topologie de niveau 2 en allant tout simplement collecter ces informations dans les équipements actifs. Après cette collecte, une corrélation des données est nécessaire pour obtenir la topologie du réseau local. Tous les algorithmes de découverte de la topologie de niveau 2 sont de ce fait assez similaires.

3.4.3.2 Point commun des algorithmes de découverte de la topologie

L'objectif est d'avoir, à partir de l'information minimale du réseau (données d'un commutateur par exemple), une topologie représentant tous les éléments qui entrent en jeu dans le fonctionnement du réseau local. Il peut s'agir des commutateurs, des serveurs, des serveurs d'impression, des postes de travail, en définitive de tout élément connecté sur le réseau local. La topologie obtenue permet d'avoir le positionnement de ces éléments par rapport à chaque commutateur, par exemple, dans l'architecture générale du réseau.

Ces algorithmes ([34], [46], [13]) utilisent tous le protocole SNMP pour collecter des informations dans les MIBs SNMP et notamment les informations contenues dans la MIB-2 Bridge qui donnent les associations `port/adresse Ethernet` des éléments connectés (cf. pour exemple la figure 3.10), appelée communément la matrice de commutation (*Forwarding Database*). En effet, la MIB Bridge contient les informations nécessaires à la commutation des trames Ethernet sur le réseau. Une utilisation des informations de la MIB Bridge est présentée plus en détail dans la section 5.4 pour la construction de la topologie d'un réseau local. En corrélant ces données, c'est-à-dire en associant les adresses Ethernet des éléments du réseau, celles découvertes dans la MIB Bridge, on peut arriver à positionner les éléments dans la topologie du réseau. Toutefois, lorsque le réseau fonctionne avec des équipements actifs non SNMP, une partie de la topologie ne peut être construite. En effet, la MIB Bridge pourra avoir enregistrée, par exemple, 30 adresses Ethernet sur le port du commutateur sans pouvoir fournir suffisamment d'information pour déterminer la topologie réelle. Dans ce cas précis, on supposera la présence d'un équipement actif (concentrateur par exemple) connectant les différents éléments.

En conclusion, cela motive le fait que tout équipement actif devrait avoir un agent SNMP. C'est d'ailleurs la tendance générale (un tel agent est présent en standard dans les commutateurs Cisco, commutateurs de plus en plus utilisés).

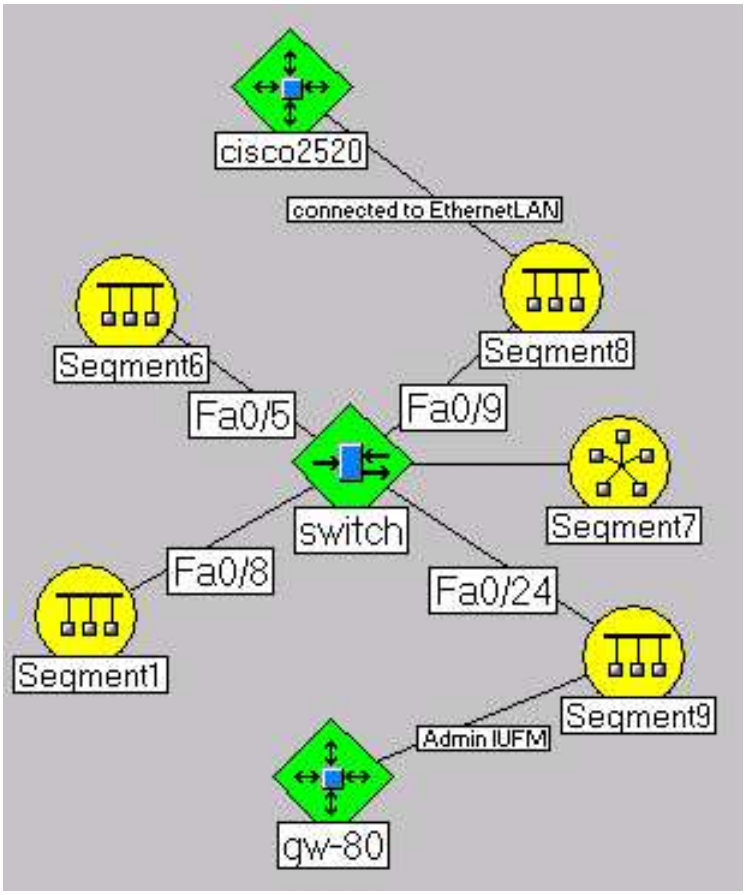

FIG. 3.10 – Topologie de niveau 2

Avec les informations collectées de plus en plus nombreuses, la topologie résultante aura une chance d'être de plus en plus précise. Cependant, il est intéressant d'avoir toute cette quantité d'information, mais il demeure néanmoins que la partie la plus importante et la plus délicate de l'algorithme de construction de la topologie reste la corrélation des données.

Nous avons été obligé d'écrire notre propre algorithme de découverte et de construction de la topologie (voir chapitre 5) parce que les algorithmes décrits précédemment ne sont pas disponibles dans le domaine public.

3.5 Bilan

Dans ce chapitre nous avons pu mettre en évidence les techniques utilisées pour déployer une plate-forme à agents mobiles : démarrage des nœuds, arrêt, redémarrage, nommage des nœuds et les techniques de localisation de ceux-ci. Après avoir évoqué les différents types d'agents mobiles qu'il est possible de mettre en œuvre, chacun ayant une fonctionnalité bien définie (agent visiteur, agent trans-

porteur, etc.), nous nous sommes penchés sur le problème de la programmation des agents mobiles dans ces plates-formes. Différentes techniques sont mises en œuvre dans les plates-formes, et nous avons vu que le schéma général reste le même : instanciation d'un agent mobile générique pour les fonctionnalités générales de l'agent mobile, puis gestion des ordres de migration de ceux-ci. Soit d'une manière manuelle (primitivie `go`), soit de manière transparente à l'agent mobile par le biais d'un système de poursuite de l'itinéraire.

Nous avons décrit les différents types d'itinéraires pouvant exister mais aussi les schémas de composition permettant d'aller au delà d'un itinéraire "séquentiel" et d'avoir des itinéraires plus structurés. Ces itinéraires permettent d'avoir des itinéraires d'administration plus évolués que le mode séquentiel.

En fin de chapitre, les outils de construction de la topologie de niveau 3 et de niveau 2 ont été vus pour mettre en évidence l'intérêt d'avoir la topologie du réseau construite automatiquement. Il s'agira d'en tirer profit pour la construction d'itinéraires d'administration système et réseau dynamique.

Nous présenterons dans le chapitre suivant la solution que nous avons mise en œuvre au dessus de la plate-forme ProActive pour fournir des itinéraires liés à l'administration système et réseau en utilisant des agents mobiles.

Chapitre 4

Conception d'un mécanisme d'itinéraires dynamiques pour l'administration système et réseau

4.1 Introduction

Nous voulons fournir un système d'itinéraires pour effectuer des tâches d'administration système et réseau. Comme étudié dans le chapitre précédent, les plates-formes à agents mobiles se fondent en général sur des itinéraires d'administration qui sont statiques (liste d'éléments dans un fichier texte par exemple) ; l'objectif est de se détacher de cette contrainte en fournissant des itinéraires dynamiques bâtis grâce à la découverte de la topologie sous-jacente du réseau à administrer. Le mécanisme d'itinéraire dynamique que nous proposons dans cette thèse est un système permettant la migration des agents mobiles en vue de l'exécution des opérations d'administration sur les différents hôtes ou éléments réseau, qui est basé sur la composition d'itinéraires dynamiques correspondant à chacun des sous-réseaux locaux constituant le domaine de l'entreprise. Nous ne fournissons pas d'itinéraires en rapport avec l'Internet ou une portion de cet Internet. En effet, il n'est pas nécessaire de prendre en compte le réseau de l'opérateur qui relie les différents sous-réseaux que nous souhaitons administrer avec la technologie des agents mobiles, parce que nous souhaitons mettre en œuvre des itinéraires d'administration dans les réseaux locaux du domaine à administrer. Ainsi, le réseau extérieur ne relève pas de notre compétence. Et donc, la topologie, même partielle du réseau IP, n'est pas prise en compte dans le cadre de notre recherche.

La section 5.4 expliquera en détail le principe et l'implantation du mécanisme de découverte de la topologie réseau du domaine à administrer.

Pour cela, nous avons utilisé la bibliothèque à agents mobiles ProActive [67] et

étendu le système existant d'itinéraire de migration pour l'adapter à un environnement d'administration système et réseau. Dans la section 4.2 nous décrirons les fonctionnalités de la plate-forme ProActive. Dans la section 4.3 nous détaillerons plus précisément le modèle de cette plate-forme dans le contexte de la migration. Les extensions apportées, sans modification du modèle de ProActive, seront présentées dans la section 4.4. Il est important de remarquer que ces extensions ont pu être faites sans modification du modèle de suivi d'itinéraire de la bibliothèque de ProActive.

4.2 Les agents mobiles dans ProActive

Les modèles à *objets actifs* proviennent de l'unification des notions d'objet et d'activité, de la même manière que le concept d'objet lui-même provient de l'unification des notions de structure de données et de procédure. Un objet actif se conforme donc à la règle des trois unités : unité de données (objets), unité de code (classes) et unité d'activité (threads).

Le concept d'*agent mobile* résulte de la rencontre entre les techniques de programmation distribuée à objets et les techniques de code mobile. Dans son acception la plus courante, la notion d'agent mobile désigne une entité logicielle autonome qui se déplace de machine en machine afin de remplir une tâche.

Même si diverses plates-formes d'agents mobiles ont été proposées (dans le cadre du langage Java, Aglets [97] et Voyager [98] pour les plus célèbres), aucune n'intègre de façon réellement transparente les aspects objets actifs communicants d'une part, mobilité d'autre part, comme réussit à le faire ProActive.

4.2.1 ProActive : objets actifs asynchrones communicants

Dans un souci de portabilité, la bibliothèque ProActive s'interdit toute modification du langage Java et de sa machine virtuelle. Nous sommes donc en présence d'une simple bibliothèque, qui utilise les outils standards (`javac` et JVM). Pour des raisons de déverminage des programmes (problème exacerbé en l'occurrence car ils sont répartis et parallèles), ProActive s'interdit également la modification du code source écrit par l'utilisateur (pas de pré-traitement). De plus, la bibliothèque est elle-même écrite entièrement en Java, ce qui autorise l'utilisation de n'importe quelle plate-forme Java et des outils associés, en particulier le fonctionnement dans un environnement ouvert et dynamique avec télé-chargement dynamique de code ; ProActive utilise le mini-serveur `http` de la plate-forme Java pour transférer dynamiquement le *bytecode* nécessaire.

4.2.2 Modèle de base

ProActive offre un modèle de programmation parallèle et répartie que l'on peut qualifier de "*à objets actifs*" [15] (par opposition à des approches où les activités sont orthogonales aux objets), et de "*hétérogène*" dans le sens où tous les objets ne sont pas actifs. Un des objectifs est de faciliter la programmation répartie, en particulier par le biais de la réutilisation [18].

En bref, le modèle de ProActive présente les caractéristiques suivantes :
- des objets actifs et accessibles à distance,
- la séquentialité des activités (processus purement séquentiels),
- une communication par appel de méthode standard,
- des appels systématiquement asynchrones vers les objets actifs,
- un mécanisme d'attente par nécessité (futur transparent),
- les continuations automatiques (un mécanisme transparent de délégation),
- l'absence d'objets partagés,
- une programmation des activités qui est centralisée et explicite par défaut,
- du polymorphisme entre objets standards et objets actifs distants.

Un *appel asynchrone* permet de ne pas se bloquer lors d'une communication. Cependant, toute communication se base sur une phase de rendez-vous (l'émetteur est bloqué jusqu'à ce que l'appel de méthode ait atteint la file des requêtes en attente de l'objet récepteur).

Un *futur* est alors un objet résultat d'un appel asynchrone : initialement il n'a pas encore de valeur. D'une façon duale, l'*attente par nécessité* permet de se bloquer automatiquement si l'on tente d'utiliser le résultat non encore revenu d'un appel asynchrone. Un futur est *transparent* dans le sens où le programmeur n'a pas besoin d'ajouter du code pour obtenir ce type de synchronisation.

Par rapport à RMI [1], ProActive se caractérise par la création d'activités à distance, par des objets distants générés à partir de classes ou d'objets (et non pas uniquement d'interfaces), par les appels asynchrones et les futurs transparents, par un contrôle fin de l'activité (file d'attente de requêtes paramétrable) des objets actifs et bien sûr la migration.

Nous allons maintenant présenter succinctement les principales caractéristiques de ProActive (pour plus de détails voir [17]).

4.2.3 Création des objects actifs

La bibliothèque a pour objectif de permettre la création d'un object actif distant le plus simplement possible, à partir de code (donc d'une classe [2]) initialement local et séquentiel. Un objet Java standard, créé par une instruction :

```
A a = new A ("foo", 7);
```

[1] Remote Method Invocation, bibliothèque standard de Java

[2] Si le code de la classe n'est pas disponible dans la JVM, il sera dynamiquement téléchargé depuis un serveur HTTP, indiqué au lancement de la JVM.

a) Instanciation-based :

```
A a = (A) ProActive.newActive ("A", params, Node);
```

b) Object-based :

```
a = (A) ProActive.turnActive (a, node);
```

FIG. 4.1 – Création des objects actifs : Instanciation-, Object-based

peut être transformé en un objet actif distant de deux manières différentes (Exemple figure 4.1).

Du plus statique au plus dynamique, **a) Instanciation-based** passe par la création d'une nouvelle classe ou d'une interface qui sera instanciée avec l'activité FIFO par défault. Si cette classe ou interface implémente l'interface-marqueur `RunActive` [3], l'activité propre de l'objet actif sera définie par la méthode runActivity, en place de l'activité FIFO réalisée par défaut (cf. section 4.2.4). Le second paramètre de cet appel statique (`params`) correspond aux paramètres effectifs à passer au constructeur lors de la création distante. Dans notre exemple, il peut être défini par :

```
Object[] params = {"foo", new Integer (7)};
```

Il est possible de redéfinir la politique de service sans que l'objet actif implémente l'interface-marqueur `RunActive` en appelant une des méthodes `newActive` de la bibliothèque, qui prend en paramètre une activité (cf. code figure 4.2).

```
A a = (A) ProActive.newActive("A",null,
"//lo.inria.fr/VM1", RunActive, null)
```

FIG. 4.2 – Création d'un objet actif avec une politique de service `RunActive`

Le dernier paramètre (`node`) spécifie la machine virtuelle où doit être placé l'objet actif (c'est habituellement une URL, par exemple, de la forme `//lo.inria.fr/VM1`). Dans le cas d'un node `null`, la création se fait dans la JVM en cours. Un autre objet actif passé en paramètre à la place du paramètre node (`newActive` est surchargée) permet la co-allocation (même machine virtuelle) avec un objet actif déjà existant (cf. figure 4.3).

Enfin, le dernier type de création, **b) Object-based**, est encore plus dynamique puisqu'il prend un objet déjà créé, et en fait un objet actif accessible à distance. La sémantique est la suivante. Tout d'abord, si le code nécessaire à un accès distant sur ce type d'objet n'existe pas encore, il est généré dynamiquement. Si `node` est la JVM en cours, les éléments nécessaires à la création d'un

[3]Interface marqueur, `RunActive` comporte une routine qui permet de redéfinir la politique de service

```
A a0 = (A) ProActive.newActive("A",params,null) ; // JVM en cours
A a1 = (A) ProActive.newActive("A",params,"//lo.inria.fr/VM1") ;
A a2 = (A) ProActive.newActive("A",params,a1) ; // co-allocation
```

FIG. 4.3 – Création d'un objet actif dans la JVM en cours, à distance ou par co-allocation

```
class BoundedBuffer extends FixedBuffer
         implements RunActive {
  runActivity (Body myBody) {
    while (true) {
      if (this.isFull()) myBody.serveOldest ("get");
      else if (this.isEmpty()) myBody.serveOldest ("put");
      else myBody.serveOldest();
      myBody.waitForNewRequest ();
} } }
```

FIG. 4.4 – Programmation explicite du contrôle

objet actif sont ajoutés à l'objet en question (une activité propre, une file d'attente, etc.). Si `node` n'est pas la JVM en cours, une copie de l'objet en question est transmise à la JVM `node`, et les éléments mentionnés ci-dessus sont ajoutés dans la JVM de cette copie. Dans tous les cas, l'objet original passif reste en place. Cette technique permet entre autre la répartition de code pour lequel on ne dispose pas du source.

4.2.4 Activités, contrôle explicite et abstractions

Les appels distants étant asynchrones, ils sont mémorisés sous la forme de *requêtes* dans une file d'attente du côté de l'appelé. Ultérieurement, nous dirons que la requête est *servie* par l'objet actif. Jusqu'à présent, nous avons créé un objet actif sans préciser son activité, et par défaut un service FIFO des requêtes était alors réalisé : exécution des requêtes dans l'ordre d'arrivée. Dans ce sens, les objets actifs sont bien des processus purement séquentiels.

La création d'objets actifs de type **a) Instanciation-based** permet de donner à un objet un comportement spécifique[4] : activité propre (qui ne consiste pas à servir des méthodes publiques), activité mixte (services et activité propre), services purs mais de type non-FIFO.

L'exemple 4.4 présente un simple tampon borné. La routine `runActivity` permet à l'utilisateur de spécifier, sous la forme d'un thread explicite, l'activité

[4]Un objet actif se situe toujours dans un continuum : *Acteur pur – Agent – Serveur pur.*

void serveOldest () ;	Sert la plus ancienne requête, bloquée
void serveOldest (String S) ;	Sert la plus ancienne de nom **S**, bloquée
void serveOldest (String S1, String S2) ;	Sert la plus ancienne de nom **S1**,**S2**
void serveOldestWithoutBlocking () ;	Service non bloquant
void serveMostRecentFlush (String s) ;	Sert la plus ancienne de nom **s**, et supprime les autres
void serveOldestTimed (int t) ;	Service bloquant pour au plus **t** ms
void waitForNewRequest () ;	Attente non active d'une requête

TAB. 4.1 – Routines de service

de l'objet. Lorsque cette routine existe, elle remplace totalement le comportement FIFO fourni par défaut. Le paramètre `myBody` donne accès à tout un ensemble de routines de services permettant de sélectionner la requête à servir. Dans l'exemple 4.4, la méthode `serveOldest` est utilisée, mais toute une bibliothèque est accessible au programmeur (exemple 4.1). On y trouve des services *bloquants*, *non-bloquants*, *temporisés*, etc. mais également des itérateurs sur la file d'attente permettant d'étendre la bibliothèque si nécessaire. Notons `waitForNewRequest` qui permet de programmer une attente non-active.

Nous sommes ici en présence d'une programmation de l'activité qui est *explicite*, avec service lui aussi explicite, sans non-déterminisme sur l'ordre des services ; dans l'exemple 4.4 si le tampon n'est ni plein ni vide, on impose de servir la requête la plus ancienne. Ce type de programmation est fort utile si l'on souhaite avoir un contrôle fin sur l'activité de l'objet.

Le fait que la programmation soit ouverte (contrôle du service des requêtes) se révèle particulièrement important pour la construction d'un service de migration puissant et flexible (voir section 4.3.2). Cet aspect de programmation ouvert, permet aisément d'étendre le modèle de ProActive pour des besoins spécifiques.

4.3 Migration

4.3.1 Migration Faible

Il est bien connu qu'il est difficile d'implémenter (c.f. [90]) de façon complètement portable de la *migration forte* en Java de par l'absence de trois fonctionnalités : interruption préemptive, lecture et sérialisation du thread, écriture (reconstruction) du thread sur la JVM cible. Par souci de portabilité ProActive s'est interdit toute modification de la machine virtuelle, toute utilisation d'outils tels qu'un préprocesseur de code source (ProActive agit donc uniquement à l'exécution) et finalement, en raison des problèmes de sécurité, toute modification des classes (le bytecode) au chargement.

```
public class SimpleAgent implements RunActive,
     Serializable {
  public void moveToHost(String t) {
      ProActive.migrateTo(t);
  }
  public void joinFriend(Object friend) {
      ProActive.migrateTo(friend);
  }
  public ReturnType foo(CallType p) {
      ...
} }
```

FIG. 4.5 – SimpleAgent : exemple d'un objet actif mobile

4.3.1.1 Primitive

Il existe en fait une seule primitive implémentée dans la bibliothèque ProActive pour fournir la migration des objets actifs communicants : `migrateTo()` . Cette primitive est surchargée en deux versions décrites dans la table 4.2.

	Permet la migration vers
`static void migrateTo(URL)`	un site référencé par une URL.
`static void migrateTo(Objet)`	le site supposé d'un autre objet actif.

TAB. 4.2 – Primitives de migration (méthodes statiques)

Un objet voulant migrer appelle lui-même la méthode statique `migrateTo()` qui se charge de toutes les opérations nécessaires à la migration, notamment la sérialisation de tout le sous-système (un objet actif et tous ses objets passifs) et sa reconstruction à l'arrivée sur le site distant. Il est possible soit de migrer vers un nœud distant que l'on désigne explicitement, soit de migrer pour rejoindre un autre agent, sans que l'on sache explicitement sur quel nœud il se trouve. Dans ce dernier cas, il n'est toutefois pas possible de garantir que, une fois la migration effectuée, les deux objets mobiles se trouveront sur le même nœud : il est possible que l'agent que l'on cherche à rejoindre migre lui aussi de site en site.

Pour utiliser cette primitive, un objet actif peut implémenter une méthode publique qui appellera la méthode statique *ProActive.migrateTo()*. Ainsi, la demande de migration pourra venir d'un objet tiers. Cependant, la décision de migrer viendra *in fine* de l'objet mobile lui-même : lors du service d'une requête portant sur une telle méthode publique. A noter que du fait de la migration faible, tout code situé après *ProActive.migrateTo()* ne sera pas exécuté. L'exemple 4.5 illustre l'utilisation des méthodes de migration dans un objet actif.

4.3.1.2 Localisation d'objets mobiles

Afin de pouvoir assurer la communication en présence de migration, il est nécessaire de pouvoir *localiser* après un nombre quelconque de migrations un objet actif. Pour être indépendant du système d'exploitation sous-jacent, la bibliothèque ProActive implémente le système de localisation dans sa couche applicative [8].

L'approche définie dans ProActive est une approche par répéteur. Plus précisément, la source d'un message n'a pas besoin de connaître la localisation exacte de l'agent mobile, il lui suffit de savoir que tout message qu'elle envoie sera correctement acheminé à destination. Pour réaliser cela, l'agent mobile qui quitte un nœud laisse derrière lui un objet répéteur qui sera chargé de faire suivre les messages vers son nouveau site d'exécution présumé (cf. figure 4.6).

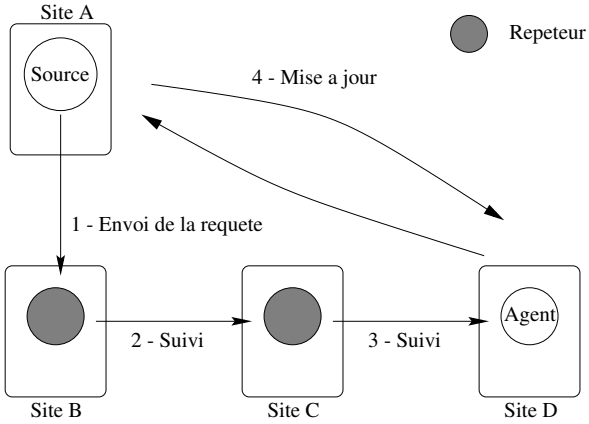

FIG. 4.6 – Localisation avec répéteurs

Ce nouveau nœud peut lui-même abriter un répéteur, et ainsi de suite pour chaque migration. On obtient ainsi une chaîne de répéteurs qui a pour dernier maillon l'objet actif mobile. Le message sera donc transmis de nœud en nœud jusqu'à l'agent. Pour améliorer la fiabilité d'un tel système de répéteurs, c'est-à-dire diminuer le risque que la chaîne des répéteurs soit brisée par une panne, ProActive implémente un système diminuant la longueur de la chaîne des répéteurs, en mettant à jour dynamiquement les références vers la nouvelle position de l'agent mobile : lorsqu'un appel de méthode atteint l'objet mobile, la localisation exacte de l'appelé est renvoyée à l'appelant. Cette mise à jour est facile à réaliser, puisque toute communication ProActive (à travers des répéteurs ou directement) se base sur la phase de rendez-vous.

4.3.2 Abstractions pour la mobilité

Le modèle et les primitives introduits précédemment permettent de construire des objets actifs mobiles en gérant explicitement les migrations. Mais il est éga-

lement possible de construire, au dessus des primitives de base, des méthodes et des concepts de plus haut niveau afin de gérer la migration à différents niveaux d'abstraction. Cela permettra entre autre de faciliter la spécification de comportements autonomes.

Dans la suite de cette section, nous allons présenter trois types d'abstractions réutilisables : l'exécution automatique de méthodes lors du départ ou de l'arrivée sur un nouveau site, le suivi d'un *itinéraire* pré-établi, et un agent mobile générique.

4.3.2.1 Exécution automatique de méthodes sur départ et arrivée

La migration d'un objet actif mobile comporte deux phases principales correspondant au déroulement nominal de la migration, ce sont le *départ* d'un site et *l'arrivée* sur le site de destination. Une troisième phase existe en présence d'une levée d'*exception*. ProActive permet de construire une abstraction qui va associer à chacune de ces phases une méthode à exécuter. Cette association se fait en appelant une des méthodes de la table 4.3 avec comme paramètre le nom de la méthode à exécuter.

	Méthode à exécuter
`static void onDeparture(String s)`	au départ
`static void onArrival(String s)`	à l'arrivée
`static void onException(String s)`	lors d'une exception

Tab. 4.3 – API d'exécution automatique de méthode, notamment sur départ et arrivée

Des contraintes ont été imposées sur les méthodes susceptibles d'être exécutées automatiquement. Elles ne doivent prendre aucun paramètre et ne doivent retourner aucun résultat. Cependant il est possible d'avoir des fonctions complexes en utilisant les attributs de l'objet. Une utilisation typique de ces méthodes est la suppression et la reconstruction automatique d'éléments non sérialisables qui ne peuvent donc pas être déplacés avec l'objet, tels qu'une interface graphique (voir exemple figure 4.7).

4.3.2.2 Itinéraire

L'autonomie d'un agent mobile provient notamment de sa capacité à aller de site en site sans intervention extérieure. Dans le cadre de la bibliothèque ProActive, un itinéraire est formé d'un ensemble de paires *destination-méthode*, instance d'une classe appelée `NodeDestination`, représentant les sites à visiter avec les actions à effectuer à l'arrivée sur chacun d'entre eux. Les hôtes à visiter (destinations au sens courant du terme) sont définis soit par reférence directe, soit

static void add(URL, String method)	ajoute un site à visiter
static void travel()	démarre le suivi d'un itinéraire
static void requestFirst(Boolean)	service des requêtes avant migration
static void itineraryStop()	arrête un itinéraire
static void itineraryResume()	reprend le suivi d'un itinéraire
static void itineraryRestart()	recommence l'itinéraire

TAB. 4.4 – API d'utilisation d'un itinéraire

```
public void deleteSwingInterface() { ... }
public void rebuildSwingInterface()  { ... }
public void migrateTo(String dest) {
  ...
  ProActive.migrateTo(dest);
}
public void runActivity(Body b) {
  ...
  b.itinerary.add(new NodeDestination("//tuba.inria.fr/Node1",
       "rebuildSwingInterface"));
  b.itinerary.add(new NodeDestination("//oasis.inria.fr/Node2",
       "rebuildSwingInterface"));
  ProActive.onDeparture("deleteSwingInterface");
  ProActive.requestFirst(true);
  ...
  ProActive.travel();
}
```

FIG. 4.7 – Exécution automatique de méthodes et itinéraires

par leur nom symbolique, auquel cas les mécanismes de nommage de RMI sont utilisés[5]. Une méthode est désignée par son nom, et est exécutée par réflexion.

Cette abstraction associe à chaque objet actif un itinéraire courant qui est suivi de manière séquentielle, et contrôlé par les méthodes de la table 4.4. Un objet mobile démarre un itinéraire en appelant la méthode `travel()`. Il est possible d'appeler la méthode `requestFirst()` pour spécifier le moment où doit intervenir le traitement des requêtes. L'agent peut ainsi suivre un itinéraire qui sera prioritaire sur le traitement des requêtes (elles seront ignorées jusqu'à la dernière destination de l'itinéraire) ou bien traiter toutes celles en attente avant de poursuivre son parcours. Il est possible de contrôler directement et dynamiquement un itinéraire afin de construire un comportement adapté à une application particulière (voir en particulier les méthodes `Stop` et `Resume` de la table 4.4).

[5]Ce qui revient à associer à chaque site une URL du type rmi ://nomdusite/nomdelobjet

```
public class GenericAgent implements Serializable {
...
 public void shareInformation() {
    int max = agentList.size();
    for (int j=0;j<max; j++) {
     ((GenericAgent)
       agentList.elementAt(j)).informationFound(. . .);
 }  }
 public void runActivity(Body myBody) {
    while (. . .) {
      this.serveAllPendingRequests();
      this.performOperation();
      this.shareInformation();
      ProActive.migrateTo(getNextDestination());
} } }
```

FIG. 4.8 – Modèle générique d'activité d'un agent mobile

4.3.2.3 Agent mobile générique

Beaucoup d'applications à base d'agents mobiles présentent les mêmes caractéristiques : un ou plusieurs agents se déplacent de site en site, effectuent sur chaque site des opérations et partagent leurs résultats. Pour décrire le fonctionnement d'un agent mobile, nous présentons sur la figure 4.8 l'activité générique d'un agent mobile qui va de site en site pour exécuter une tâche T et qui partage l'information recueillie avec d'autres agents mobiles du même type. Cet agent mobile générique est réalisé en utilisant la bibliothèque ProActive et montre succintement comment est fait l'implémentation de la migration et comment est réalisé le partage d'information.

L'activité de l'objet (cf. figure 4.8), se décompose en plusieurs opérations. L'agent sert d'abord toutes les requêtes en attente, puis il effectue une opération (par exemple une recherche d'information). L'information est ensuite partagée avec les autres objets mobiles. Le code nécessaire pour cette tâche est extrêmement simple puisqu'il consiste à appeler une méthode sur chacun des autres objets, le moteur d'exécution ProActive se chargeant de transformer l'appel local en appel distant quelque soit le site sur lequel se trouve le destinataire.

Il suffit de redéfinir la méthode *performOperation()* afin de programmer les fonctions précises de l'agent. Il est aussi possible d'avoir une politique de sélection de site (méthode *getNextDestination()*) totalement aléatoire ou bien dépendante des informations déjà recueillies.

4.3.3 Modèle de suivi d'itinéraire de ProActive

La gestion du parcours des Destinations d'un itinéraire (génération des ordres de migration) et le lancement des fonctions à exécuter au cours du suivi de cet itinéraire est programmée dans une classe de ProActive, nommée `MigrationStrategyManager`, et une instance de cette classe est associée à l'objet. Lorsque l'itinéraire de migration est prêt, c'est-à-dire lorsque le gestionnaire du suivi de l'itinéraire a connaissance de tous les nœuds à visiter, alors le `MigrationStrategyManager` parcourt successivement chaque élément de cet itinéraire. Pour chaque élément il exécute l'appel de la méthode associée (voir section 4.3.2.1) en fonction du paramètre associé à la Destination, c'est-à-dire à l'arrivée sur le nœud (par exemple `onArrival`) ou au départ (par exemple `onDeparture`) du nœud ou la combinaison des deux. Ainsi, le `MigrationStrategyManager` décharge l'agent mobile du suivi de son propre itinéraire. Lorsque tous les éléments de l'itinéraire ont été passés en revue, la migration de l'agent mobile s'arrête.

La figure 4.9 présente un exemple d'itinéraire qu'un agent mobile peut exécuter en utilisant la bibliothèque ProActive. L'agent mobile part du nœud Thio pour se rendre successivement sur les nœuds Yate, Koumac et Noumea pour effectuer la fonction `onArrival` (par exemple). En résumé, cet itinéraire consiste à réaliser dans l'ordre :

- Départ de la station d'origine
- Pour chaque nœud de l'itinéraire exécuter l'action *onArrival*, à l'arrivée sur celui-ci
- Passer au nœud suivant et recommencer

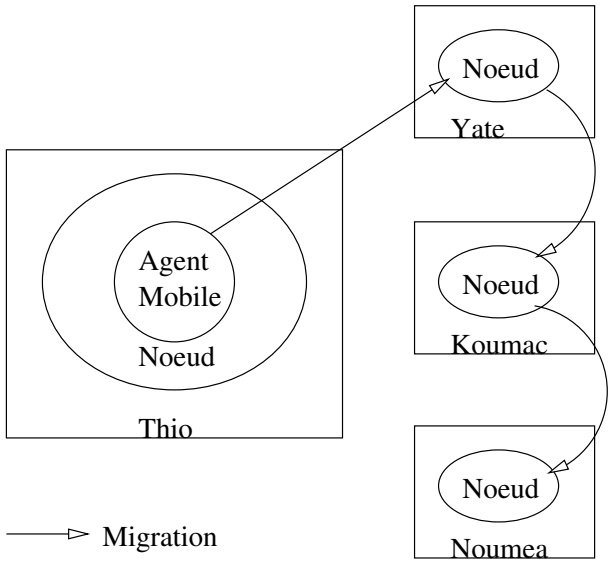

FIG. 4.9 – Un itinéraire défini manuellement, construit avec la bibliothèque ProActive

Nous explicitons dans les sections suivantes comment nous avons utilisé le concept de `Destination` et les extensions que nous y avons apportées afin de pouvoir construire des itinéraires d'administration système et réseau.

4.4 Conception d'itinéraires pour l'administration

Un itinéraire d'administration est un itinéraire qui permet à un agent mobile d'effectuer des tâches à la place d'un administrateur. Nous regroupons dans le concept d'itinéraire d'administration les opérations qui peuvent être effectuées sur les systèmes d'exploitation via des nœuds de ProActive et les opérations d'administration réseau qui sont effectuées selon la technologie Client/Serveur SNMP.

Tout en restant fidèle au modèle de ProActive, nous avons étendu la définition de ce qu'est une `Destination` pour que cela inclut les équipements actifs du réseau pour lesquels une opération d'administration doit être effectuée. Cette opération est une opération qui doit être réalisée en utilisant le protocole SNMP afin d'accéder aux informations de la MIB de chaque agent SNMP.

4.4.1 Destinations

4.4.1.1 Les types d'intervention d'un agent mobile d'administration

Nous découpons les types d'intervention d'un agent mobile en quatre catégories, identifiables par leur mode de fonctionnement.

1. L'agent mobile effectue une migration sur un nœud afin d'y effectuer une tâche (1^{er} cas sur la figure 4.10)
2. L'agent mobile doit se comporter comme un client d'un agent SNMP pour collecter des données de la MIB SNMP ($2^{ème}$ cas sur la figure 4.10)
3. L'agent mobile se rapproche au plus près de l'agent SNMP en effectuant au préalable une migration sur le site qui héberge l'agent SNMP. La communication Client/Serveur utilise l'interface de bouclage (*loopback*) de la couche IP ($3^{ème}$ cas sur la figure 4.10)
4. L'agent mobile se rapproche des agents SNMP en effectuant une migration sur un nœud appartenant au même réseau que ces agents. Toutes les requêtes Client/Serveur sont faites en utilisant le réseau local, plutôt que les liens inter-réseaux. Ainsi les requêtes peuvent se faire à la vitesse des liens locaux ($4^{ème}$ cas sur la figure 4.10)

Ces différents types d'intervention seront entièrement définis dans l'itinéraire d'administration qui sera fourni à l'agent mobile. On trouve donc dans les itinéraires d'administration des mécanismes permettant la migration pour les agents

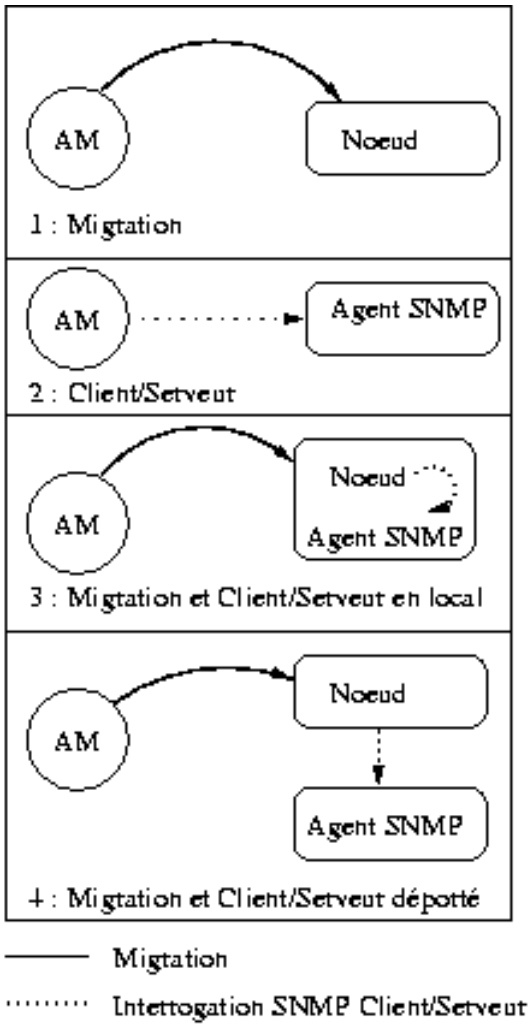

FIG. 4.10 – Types d'intervention d'un agent mobile d'administration

mobiles (en particulier la définition des nœuds d'accueil) et des mécanismes permettant d'interroger des agents SNMP du réseau.

4.4.1.2 Extension de la notion de Destination

Pour prendre en compte dans nos itinéraires les éléments compatibles SNMP, nous avons défini une Destination particulière dénommée `SNMPDestination` conforme au modèle des `Destinations` de ProActive. Une `SNMPDestination` contient l'ensemble des informations nécessaires pour effectuer une administration à distance (Client/Serveur) entre le nœud sur lequel est situé l'agent mobile et l'agent SNMP cible. On y mémorise le nom logique de l'agent SNMP cible (par exemple rt-nat-192 ou à défaut son adresse IP), le nom de la communauté en lecture, le nom de la communauté en lecture/écriture et le port UDP de l'agent SNMP. Une

`SNMPDestination` contient donc toutes les informations nécessaires pour dialoguer avec un équipement actif. Toutefois, une `SNMPDestination` ne permet pas à un agent mobile de migrer vers l'élément du réseau concerné puisque, les équipements actifs n'intègrent pas de nœuds Java pour accueillir des agents mobiles.

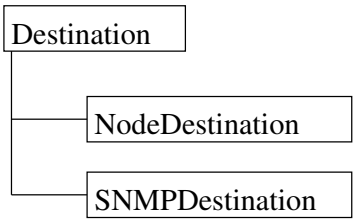

FIG. 4.11 – Hiérarchie de classes issue de la classe Destination

Par héritage de la classe Destination (cf. figure 4.11), on pourrait facilement décrire un nouveau type de Destination, comme par exemple la destination `SNMPV3Destination` pour supporter le protocole SNMP V3 ou une destination pour le protocole CMIP [104]. Pour effectuer des tâches en parallèle sur des éléments constituant le réseau, la création d'agents mobiles effectuant la même fonction peut s'avérer nécessaire (cf. section 3.3.3). Un agent mobile père créérait des agents mobiles fils, et attendrait la synchronisation de ses fils pour collecter le compte rendu de l'exécution de la fonction. Ce type d'itinéraire peut être mis en œuvre par un nouveau type de destination (par exemple `CloneDestination`) qui clônerait un agent mobile et qui permettrait la synchronisation de tous les agents mobiles fils après la collecte de l'information (par exemple `RendezVous-Destination` pour la synchronisation à la fin des tâches).

La figure 4.12 montre un itinéraire d'administration générique réalisé avec notre extension de ProActive. Il fonctionne comme suit :

- Contacter un serveur afin de récupérer un itinéraire contenant la liste des éléments à visiter
- Pour chaque élément de l'itinéraire faire
  - Cas de
    - `NodeDestination` alors migrer et exécuter l'action *onArrival* en tant que fonction d'administration
    - `SNMPDestination` alors contacter l'agent SNMP et effectuer l'opération d'administration réseau (en mode Client/Serveur local ou distant)
  - Fin de Cas
- Fin Pour

La table 4.5 rappelle brièvement les actions à mener en fonction du type de Destinations que nous venons de décrire.

En utilisant ces nouvelles définitions, les itinéraires d'administration pourront être constitués d'éléments de type `NodeDestination`, de type `SNMPDestination`, selon le type d'administration que l'on veut faire exécuter à un agent mobile d'administration.

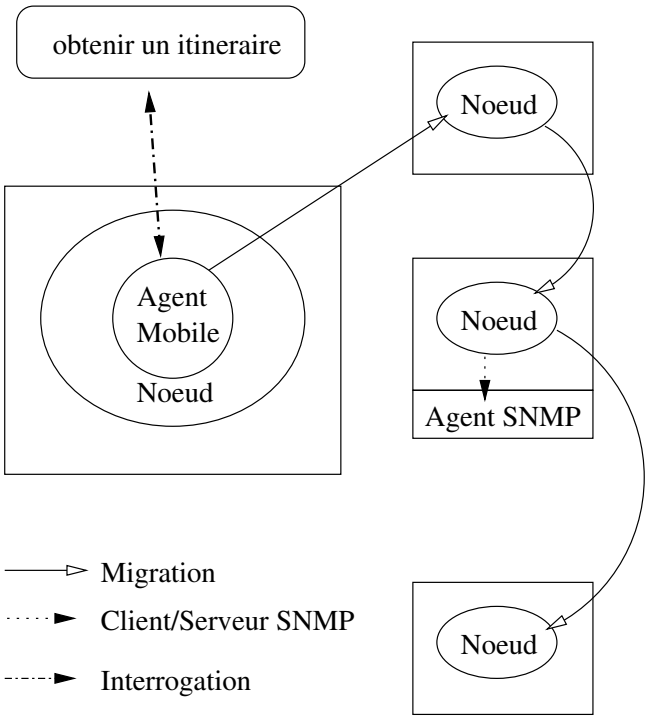

FIG. 4.12 – Un itinéraire dynamique construit selon le modèle des Destinations

Destination	Action
NodeDestination	Migration et exécution de l'appel d'une méthode
SNMPDestination	Appel de méthode pour un accès en Client/Serveur SNMP

TAB. 4.5 – Actions menées selon les types de Destination

4.4.2 Techniques d'obtention des éléments d'un itinéraire

Nous rappelons que nous sommes supposés effectuer de l'administration système et réseau sur plusieurs éléments. Tous ces éléments ne sont pas forcément dans le même sous-réseau, ce qui va nécessiter des techniques appropriées pour mettre l'information concernant les éléments du réseau à administrer à disposition. Nous présentons ci-après les techniques que l'on pourrait utiliser pour la mise à disposition des listes d'éléments permettant de construire des itinéraires, et le modèle que nous avons choisi d'implémenter.

4.4.2.1 Serveur d'annuaire centralisé

L'itinéraire est construit à partir de la liste des éléments à visiter stockés dans un serveur d'annuaire (par exemple LDAP [44], cf. figure 4.13). Les informations décrivant les nœuds de la plate-forme ou les agents SNMP des équipements actifs doivent être enregistrées dans ce serveur d'annuaire. Il en est de même lorsque des changements de topologie interviennent au sein du réseau de l'entreprise, ou lors de l'apparition ou de la disparition de nœuds. Afin de savoir où aller, l'agent mobile consulte cet annuaire, récupère la liste des éléments à visiter, construit son itinéraire, et exécute la ou les fonctions d'administration programmées par l'administrateur.

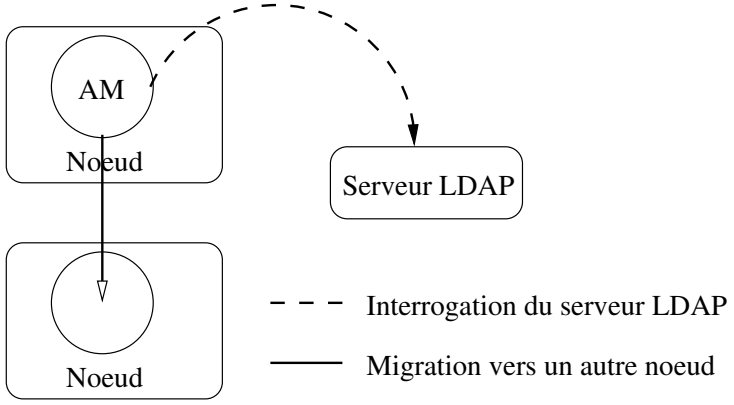

FIG. 4.13 – Serveur d'annuaire contenant les éléments nécessaires à la création des itinéraires d'administration

Dans cette proposition, chaque agent mobile doit connaître la localisation du serveur d'annuaire dans le réseau de l'entreprise. Chaque nœud de la plate-forme à agents mobiles doit s'enregistrer automatiquement et se désenregister à la fin de son activité. Si un nœud vient à disparaître subitement, un élément de la plate-forme à agents mobiles doit assurer la cohérence des enregistrements de l'annuaire en effaçant l'enregistrement correspondant au nœud défaillant. Toutefois un avantage fourni par un serveur d'annuaire LDAP est la possibilité de réplication. Ainsi plusieurs annuaires peuvent être répartis sur l'ensemble du réseau, chacun étant responsable du sous-réseau dans lequel il fournit son service, mais pouvant servir d'annuaire de secours.

4.4.2.2 Distribution d'itinéraires par un object actif

La liste des éléments permettant de construire un itinéraire à la volée est mise à disposition par un objet actif que l'agent mobile interroge avant chaque migration. Cet objet actif sert d'interface entre les agents mobiles et un système quelconque permettant d'obtenir une liste d'éléments (par exemple une base de données, un fichier décrivant les éléments, etc). L'objet actif fournit à chaque

demande de l'agent mobile le prochain élément que l'agent mobile doit visiter. L'agent mobile doit connaître la référence de cet objet actif pour le localiser et l'interroger (cf. figure 4.14). Chacun des nœuds de la plate-forme doit être enregistré dans cet objet actif (ou sa référence annulée en cas de défaillance) afin d'être accessible pendant le suivi de l'itinéraire.

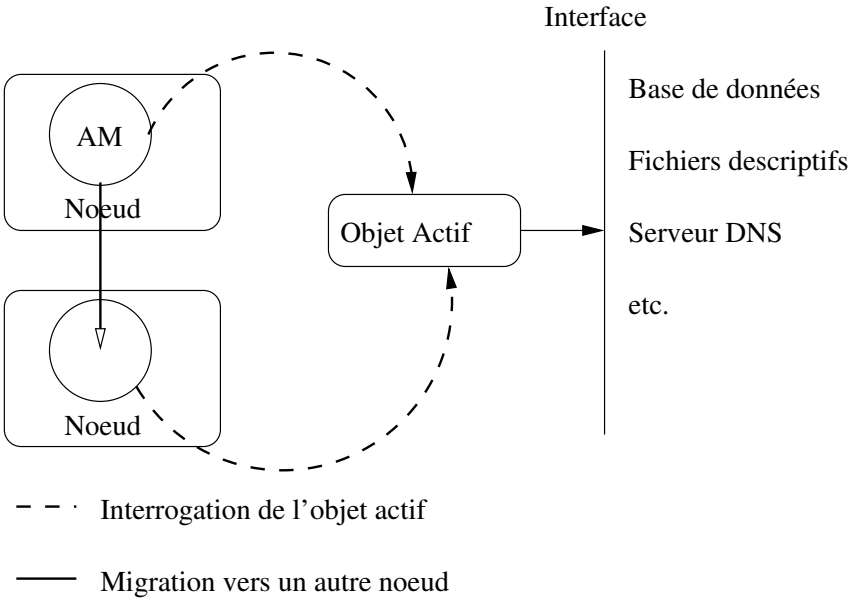

FIG. 4.14 – Objet actif fournissant des itinéraires pas à pas

Cet objet peut agir comme un *daemon de routage*. Dans ce cas, cet objet sélectionne en fonction de la demande d'un agent mobile, le prochain site à visiter selon des critères pré-définis (charge réseau, charge machine, événement transmis par d'autres agents mobiles, etc..) afin que l'agent mobile puisse continuer sa tâche. De même, les agents SNMP des équipements actifs du réseau doivent être référencés si ceux-ci doivent être administrés par un agent mobile. En cas de nœud défaillant, l'agent mobile interroge de nouveau l'objet actif pour connaître le prochain site à visiter. Il en est de même si un agent SNMP ne peut être contacté par l'agent mobile.

Dans cette proposition, la création et la gestion de l'itinéraire sont délégués à l'objet actif. L'agent mobile est un client de l'objet actif qui lui fournit au fur et à mesure le prochain site qu'il doit prendre en compte. Il est indispensable d'assurer la fiabilité de cet objet actif puisqu'il est responsable de la fabrication de l'itinéraire et de la progression du suivi de l'itinéraire par l'agent mobile. Bien évidemment, chaque consultation de l'object actif, avant toute nouvelle migration de l'agent mobile, engendre un délai dans l'exécution globale de l'itinéraire d'administration. Toutefois, la proposition de l'objet actif présente un intérêt puisque elle rend inutile la mise en œuvre de nouveaux services sur le réseau (principalement pas de service d'annuaire à mettre en œuvre). L'objet actif peut être défini

dans le même langage de programmation que la plate-forme à agents mobiles et réagir rapidement sur l'itinéraire donné à l'agent mobile lorsque des nœuds sont défaillants. De plus, l'objet actif peut utiliser les informations sur le comportement du réseau afin d'adapter les itinéraires qu'il donnera aux agents mobiles au moment de leur requête.

4.4.2.3 Critères de sélection

Les deux propositions présentées mettent en évidence des contraintes d'utilisation et de délai d'exécution. En effet la mise en œuvre d'un annuaire LDAP entraîne automatiquement une programmation statique de la localisation de cet annuaire sur le réseau (par exemple nom DNS du serveur qui héberge l'annuaire). Il en est de même pour l'object actif de la deuxième proposition. Nous pouvons dégager de ces deux propositions les propriétés suivantes qu'il est souhaitable de mettre en œuvre :

- répartir le service sur un ou plusieurs sous-réseaux
- avoir des informations régulièrement mises à jour
- éviter les références statiques pour l'accès à ces services
- diminuer le nombre de consultations du service fournissant les listes d'éléments

En prenant en compte ces besoins, nous avons donc implémenté notre propre service d'itinéraire.

4.4.3 Le modèle de service d'itinéraire proposé

4.4.3.1 Présentation globale

Ainsi, nous avons choisi d'implémenter un modèle qui est une variante des deux solutions proposées précédemment. Dans le modèle adapté (cf. figure 4.15), nous faisons transporter à l'agent mobile toutes les données qui lui sont nécessaires. L'objet actif, qui fournit la liste des éléments, est consulté une fois avant que l'agent mobile démarre son itinéraire. L'objet actif a la charge de fournir des informations les plus à jour possible aux agents mobiles, en collectant les données obtenues par un algorithme de découverte de la topologie (voir la section 5.4 pour les détails). L'objet actif est construit dans le même language de programmation que la plate-forme à agents mobiles ce qui favorise la compatibilité entre les objets de la plate-forme.

En donnant des listes d'éléments complètes aux agents mobiles un gain de temps appréciable est réalisé pour le déroulement du suivi de l'itinéraire. Cependant, si un des nœuds devient défaillant, l'agent mobile progressera sur le suivi de son itinéraire en ne tenant pas compte du nœud défaillant. Une possibilité peut être offerte aux agents mobiles afin de prévenir le service d'itinéraire d'un ou plusieurs nœuds défaillants (utilisation de la méthode `onException`, cf. table 4.3).

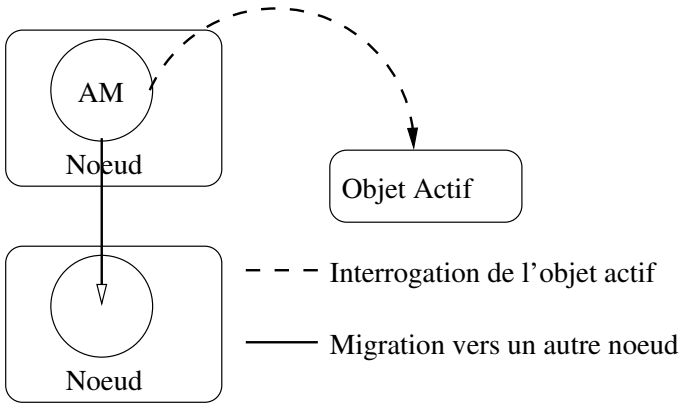

FIG. 4.15 – Objet actif fournissant des itinéraires en une seule fois

Le nœud défaillant pourra faire l'objet d'une intervention de l'administrateur.

Pour ne pas avoir de références statiques vers l'objet actif en charge de fournir la liste composant les éléments du réseau à administrer, nous avons utilisé le service d'enregistrement de Jini [53].

Pré-requis : Pour utiliser Jini dans cette architecture, cela impose le passage du trafic multicast entre les différents sous-réseaux[6] de l'entreprise utilisant une telle plate-forme.

En effet Jini offre un service d'enregistrement pour les objets qui fournissent un service sur le réseau. Une explication plus approfondie de Jini et son adaptation à notre environnement est présentée en annexe 9.1. Le service d'enregistrement (`Lookup Service`) utilise les canaux multicast (adresse IP de classe D du type `224.0.0.0`) afin de pouvoir être localisé sur le réseau. Le service de localisation de Jini (mise en œuvre par le `Lookup Service`) utilise les mêmes canaux multicast pour localiser le service d'enregistrement et obtenir la référence du service sur le réseau.

En utilisant Jini, nous sommes en mesure de détacher tous les agents mobiles des références statiques qui ont été mises en évidence dans la section 4.4.2. Ansi, chaque agent mobile pourra contacter directement le service du réseau lui offrant la liste des éléments qui sont potentiellement administrables.

4.4.3.2 Définitions d'itinéraires de visite séquentielle

En possession de ces listes d'éléments, plusieurs types d'itinéraires deviennent alors possible pour un agent mobile. On peut les classer en deux grandes familles,

[6]Dans la communauté de recherche RENATER, le trafic multicast est de plus en plus souvent ouvert et donc Jini peut s'utiliser sur cette infrastructure.

ceux qui sont ciblés et ceux qui sont typés.

Itinéraire ciblé : un itinéraire ciblé est un itinéraire dont les éléments de Destination n'appartiennent qu'à un sous-réseau. La discrimination des éléments de l'itinéraire se fait par rapport à l'adresse du sous-réseau cible.

Itinéraire typé : un itinéraire typé est un itinéraire regroupant un ensemble d'éléments ayant des propriétés similaires, comme par exemple un ensemble d'hôtes ayant le même type de système d'exploitation.

Par exemple, en itérant sur ces définitions d'itinéraires, on peut dire qu'un itinéraire général, qui permet le parcours de tout le réseau, est l'union de l'ensemble des itinéraire ciblés appliqués à chaque sous-réseau de l'entreprise. Cet itinéraire général comportera donc tous les éléments de type nœuds et la liste de tous les équipements actifs du réseau.

4.4.3.3 Description du service d'itinéraire

Nous avons défini un objet actif appelé l'`ItineraryServer`. Le rôle de cet objet actif est de mettre à la disposition des agents mobiles la liste des éléments représentant les nœuds de la plate-forme et la liste des équipements actifs. Cette liste d'éléments est obtenue à partir de l'algorithme de construction de la topologie du réseau à administrer, présentée dans le chapitre 5. Chaque `ItineraryServer` s'enregistre automatiquement en tant que service Jini afin de pouvoir être localisé par les agents mobiles. Ainsi, un `ItineraryServer` joue en quelque sorte le rôle de l'objet actif proposé dans la section 4.4.3 (figure 4.16).

Pour adapter notre modèle à l'administration système et réseau nous devons tenir compte de l'existence de sous-réseaux dans le réseau d'une entreprise. Pour s'adapter à l'existant, notre modèle sera répliqué dans chaque sous-réseau afin de fournir localement les listes des éléments locaux représentant les nœuds et les équipements actifs du réseau. Comme nous allons le voir, les serveurs d'itinéraires de ces sous-réseaux seront utilisés successivement par un agent mobile si besoin.

4.4.3.4 Destination "service d'itinéraire"

Partant du fait que nous souhaitons administrer plusieurs sous-réseaux, nous avons introduit une nouvelle Destination appelée `ISDestination`. Cette destination dans le modèle des `Destinations` (cf. figure 4.17) permettra le passage d'un sous-réseau à un autre sous-réseau. Cette `Destination` contient la référence sous forme d'URL d'un `ItineraryServer` et le nom de la méthode permettant d'enrichir l'itinéraire courant. Ainsi, un tel itinéraire intégrant ce nouveau type de `Destination` permettra à un agent mobile de passer d'un sous-réseau à un

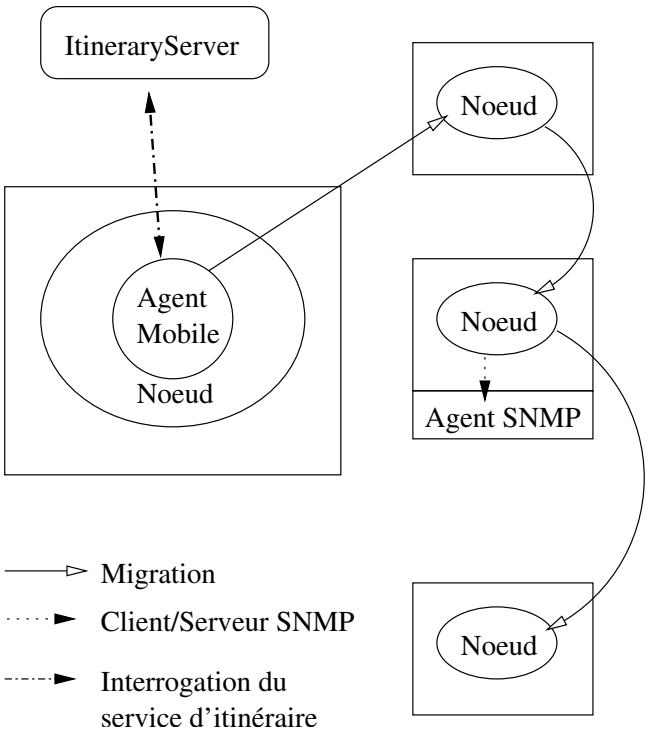

FIG. 4.16 – Le service d'itinéraire

autre de manière complètement transparente et de progresser dans le suivi de son itinéraire.

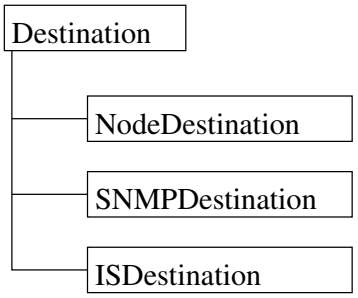

FIG. 4.17 – Extension de notre hiérarchie des `Destinations`, incluant la notion de service d'itinéraire

4.4.3.5 Destination de synchronisation

Il n'existe pas que des itinéraires de type séquentiel, mais nous pouvons proposer simplement, par extension de la notion des `Destinations`, des itinéraires permettant aux agents mobiles de fonctionner en parallèle. L'intérêt est de pouvoir faire exécuter la même tâche à des agents mobiles sur des éléments du réseau

(cf. section 3.3.3).

Itinéraire parallèle : Un itinéraire permettant à des agents mobiles d'exécuter des tâches en parallèle est un itinéraire qui imposera à l'agent mobile principal la création d'un ou plusieurs agents mobiles secondaires. Chaque sous itinéraire est un itinéraire qui peut être séquentiel (cf. section 4.4.3.2), ou alors lui-même un nouvel itinéraire parallèle. Ce type d'itinéraire nécessite une synchronisation partielle ou complète des données collectées par les agents mobiles secondaires.

Pour réaliser ce type d'itinéraire fonctionnant en mode parallèle, nous avons étendu notre définition des itinéraires pour y intégrer le concept de deux nouvelles destinations (cf. figure 4.18) qui sont :

- une `CloneDestination` qui permet de créer des agents mobiles pour un fonctionnement en parallèle
- une `RendezVousDestination` qui permet de synchroniser, si nécessaire, les résultats de ces agents mobiles

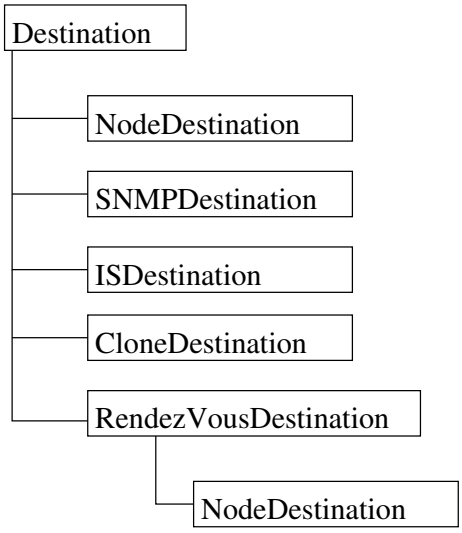

FIG. 4.18 – Extension de la hiérarchie des destinations, pour les destinations de type parallèle

La destination `CloneDestination` permet de bloquer le déroulement de l'itinéraire de l'agent mobile concerné et de lui permettre, soit de se cloner, soit de créer des agents mobiles secondaires qui exécuteront une tâche déterminée. Cette destination met à la disposition de l'agent mobile le nombre d'agents mobiles secondaires qu'il doit créer.

Pour synchroniser le travail des agents mobiles nous utilisons la destination `RendezVousDestination`. Cette destination est une destination qui englobe une `NodeDestination`. La `RendezVousDestination` peut être considérée comme la destination finale de l'itinéraire des agents mobiles secondaires puisqu'ils sont en

mesure de retourner les informations du travail accompli. Le processus de synchronisation n'impose pas que tous les agents mobiles secondaires aient terminé leur tâche pour que l'agent mobile principal puisse poursuivre la sienne. Toutefois, une synchronisation complète est possible sur le même schéma. Ce type de synchronisation dépend effectivement de l'implémentation de l'agent mobile.

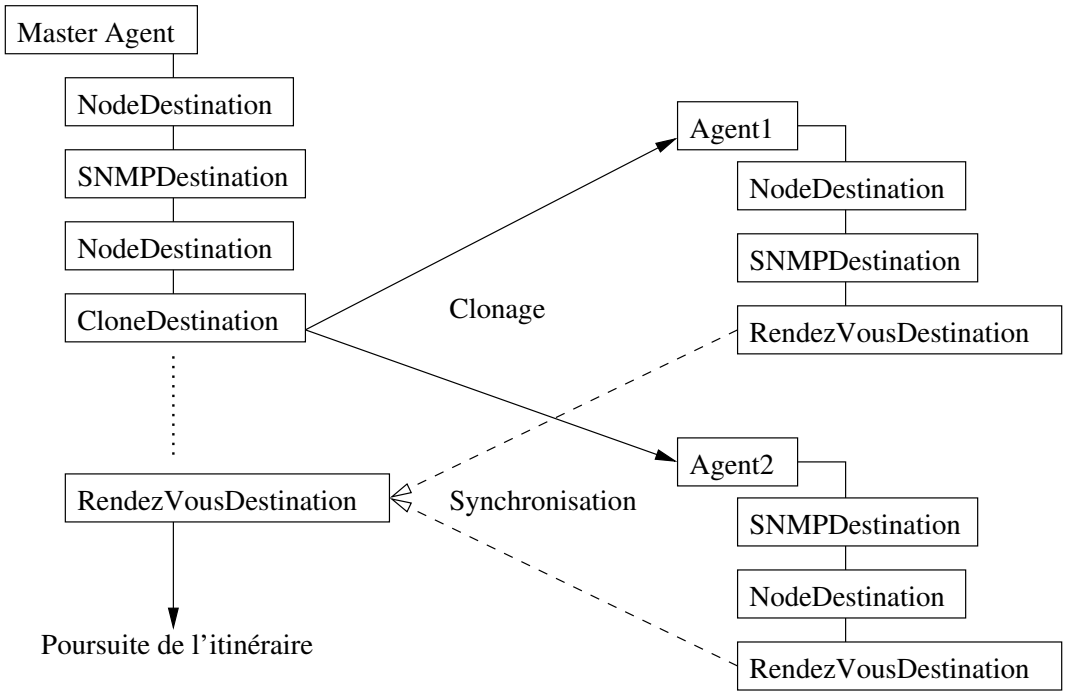

FIG. 4.19 – Exemple d'intégration de `CloneDestinations` et de `RendezVous-Destinations` dans un itinéraire de type parallèle

Par exemple (cf. figure 4.19), basons-nous sur un agent mobile qui souhaiterait connaître la charge réseau de N systèmes. Dans son itinéraire de visite séquentielle, il sera interrompu par une `CloneDestination`. A partir de cette destination, il créera N agents mobiles secondaires ayant pour tâche la collecte de la charge de chaque système que devait visiter l'agent mobile principal. L'agent mobile principal sera en attente du résultat de tous les agents mobiles secondaires. Dans ce cas précis, l'agent mobile principal continuera sa tâche uniquement lorsque tous les agents mobiles secondaires seront venus se synchroniser avec lui. Nous utilisons donc une synchronisation complète de tous les agents mobiles.

Pour illustrer un mode de synchronisation avec un seul agent mobile secondaire, prenons l'exemple de la recherche d'un espace disque suffisant pour sauvegarder des données sur N systèmes. La procédure est exactement la même que dans l'exemple précédent, à la différence que l'agent mobile principal continuera sa tâche dès qu'il aura obtenu une réponse positive de l'un de ses agents mobiles secondaires, ou alors l'ensemble des réponses ne pourra satisfaire la demande de

l'agent mobile père, ce qui sera un échec de la recherche.

Ainsi pour créer de nouveaux types d'itinéraires intégrant de nouvelles destinations, il suffit d'étendre la hiérarchie de la classe `Destination` pour mettre en œuvre les nouveaux types nécessaires aux opérations d'administration système et réseau.

4.4.4 Construction et gestion de l'itinéraire

Sur une architecture où nous regroupons des listes d'éléments différents (nœuds et équipements actifs), il est concevable de déléguer à un gestionnaire la construction d'un itinéraire. Cela permet de s'affranchir de la création systématique de l'itinéraire et d'offrir, en quelque sorte, des itinéraires "prêts à l'emploi" aux développeurs d'agents mobiles d'administration. Pour réaliser cela, nous avons créé la classe abstraite `ItineraryManager` [73] qui définit les tâches minimales que doit offrir le gestionnaire de l'itinéraire. Les méthodes fournies par cette classe regroupent : la localisation du service Jini du ou des `ItineraryServers` ; la préparation de l'itinéraire ; l'exécution du suivi de l'itinéraire ; l'arrêt du suivi de l'itinéraire ; l'ajout en cours d'itinéraire d'une `Destination` urgente ; permettre à l'agent mobile de savoir à tout instant sur quelle `Destination` il effectue une tâche.

En définissant ainsi notre gestionnaire d'itinéraire, nous pouvons construire des sous-classes de l'`ItineraryManager` qui permettent de typer les itinéraires selon la tâche d'administration à effectuer (cf. figure 4.20 et figure 4.21). Deux grandes familles générales d'itinéraires existent. Celle dont les itinéraires seront déroulés de manière totalement séquentielle que nous présentons dans la section 4.4.4.1.

Ce sont les itinéraires les plus simples à mettre en œuvre tant du point de vue du concepteur que de l'utilisation par le programmeur. La deuxième famille est celle des itinéraires dont les éléments font intervenir des états particuliers : Duplication, Point de Rendez-Vous, Choix Sélectif parmi plusieurs `Destination` possible, en sont des exemples. Cette famille d'itinéraires sera présentée dans la section 4.4.4.2.

4.4.4.1 Itinéraire de visite séquentielle

Nous avons donc défini des itinéraires de visite séquentielle suivant :
- une administration complète du réseau local (ItineraryManagerLocalNetwork, ItineraryManagerLocalNodes),
- une administration distante, dans un sous-réseau ciblé (ItineraryManagerFilterNetwork),
- une administration incluant tous les sous-réseaux :
  - tous les nœuds (ItineraryManagerAllNodes),

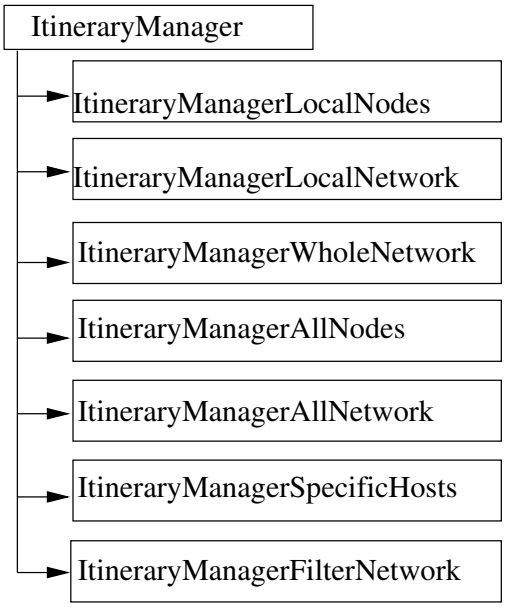

FIG. 4.20 – Hiérarchie de classes issue de la classe `ItineraryManager`

  - tous les éléments SNMP (ItineraryManagerAllNetwork),
  - tous les nœuds et tous les éléments SNMP (ItineraryManagerWholeNetwork),
- une administration incluant tous les éléments de même type (ItineraryManagerSpecificHosts) :
  - toutes les imprimantes,
  - tous les routeurs,
  - tous les commutateurs
  - etc...

Chacune des classes d'itinéraires définies ci-dessus construisent un itinéraire adapté à chaque situation. Pour diversifier les types d'itinéraires de la plate-forme, il suffit de dériver une nouvelle classe, c'est-à-dire une nouvelle façon de visiter certains éléments du réseau.

4.4.4.2 Itinéraire de visite parallèle

Les itinéraires qui entraînent des visites en parallèle sont des itinéraires où une portion de l'itinéraire sera de type séquentiel. Cependant, la partie la plus intéressante, reste la mise en œuvre du gestionnaire d'itinéraire. Par rapport à la hiérarchie de la classe `ItineraryManager` une extension donne un nouveau type de gestionnaire que nous avons mis en œuvre : l'`ItineraryManagerClone` (cf. figure 4.21). L'`ItineraryManagerClone` est un gestionnaire qui va permettre à l'agent mobile principal (voir les exemples section 4.4.3.5) de se cloner ou de créer des agents mobiles secondaires pour effectuer une tâche d'administration

système et réseau en parallèle sur plusieurs systèmes simultanément.

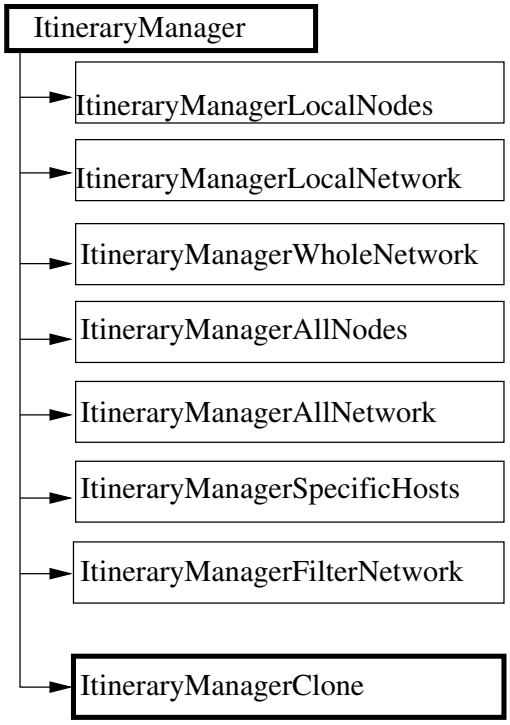

FIG. 4.21 – Hiérarchie de la classe ItineraryManager étendue pour les itinéraires parallèles

L'`ItineraryManagerClone` offre des fonctionnalités plus spécifiques que l'ItineraryManager. Il créé à la volée, un ou plusieurs sous-itinéraires de l'itinéraire que l'agent mobile doit suivre, afin de répartir dans chaque sous-itinéraire une partie des `Destinations`. L'`ItineraryManagerClone` évite par cette procédure le recouvrement de mêmes `Destinations` dans les sous-itinéraires. L'`ItineraryManagerClone` fixe la destination `RendezVousDestination` des sous-itinéraires pour que le ou les agents mobiles d'administration puissent se synchroniser, si nécessaire, avec l'agent mobile père, fournissant ainsi le résultat de leur travail.

Par extension de la classe `ItineraryManager`, on obtient un nouveau type de gestionnaire d'itinéraire adapté à des parcours en parallèle des agents mobiles d'administration système et réseau.

4.4.5 Utilisation d'itinéraires

Dans la bibliothèque ProActive la classe `MigrationStrategyManager` est fournie et a pour rôle de réaliser le suivi d'un itinéraire composé uniquement de `NodeDestination` (voir section 4.3.3). La classe `MigrationStrategyManager` prend totalement en charge le suivi de l'itinéraire et les appels de méthodes qui

sont associées à chaque `Destination`. La classe `MigrationStrategyManager` sert avant toute migration, les requêtes qui pourraient être en attente dans la file d'attente des requêtes de l'agent mobile. Une instance de cette classe doit être affectée à chaque objet actif mobile.

Pour prendre en compte de nouvelles formes de Destinations dans le modèle de ProActive, nous avons étendu cette classe pour que les actions à entreprendre en fonction du type de Destinations soient gérées (cf. figure 4.22). Étant données nos extensions sur les `Destinations` (cf. figure 4.18), il s'agit d'adapter le comportement de l'agent mobile à ces nouvelles `Destinations`. Ainsi, la migration n'est effective que pour les Destinations `NodeDestination` (pour la partie administration), `ISDestination` (pour la partie enrichissement de l'itinéraire), `CloneDestination` et `RendezVousDestination` pour la partie liée à la duplication des agents mobiles. Quant à la `SNMPDestination`, aucune migration n'est requise (modèle Client/Serveur en SNMP).

Quel que soit le véritable type des `Destinations`, cela engendre toutefois l'appel d'une méthode pour lancer l'exécution de la tâche associée. Par exemple, une Destination de type `SNMPDestination` engendrera l'appel de la méthode associée (que l'on nomme typiquement *SnmpOnArrival*) pour une opération d'administration en Client/Serveur.

La destination `ISDestination` obligera l'agent mobile à contacter le prochain `ItineraryServer` situé dans la liste des `ItineraryServers` obtenue au préalable auprès d'un Lookup Service, pour compléter son itinéraire d'administration.

```
Tant Que Itineraire non Fini Faire
  Tant qu'il y a des appels de methode en attente dans la file d'attente faire
     Servir les requêtes
  Fin Tant Que

  Considérer la prochaine Destination
   Devons-nous miger ?
       non : Appel de la méthode associée à la destination (SNMPDestination)
       oui : Si NodeDestination alors l'agent mobile migre et
                on exécute l'appel de la méthode à l'arrivée
             Si ISDestination alors on contacte l'ItineraryServer associé pour
                enrichir l'itinéraire de migration courant
             Si CloneDestination alors
                appel de la méthode permettant de dupliquer l'agent mobile et
                de préparer le point de rendez-vous qui suivra
             etc .... (par exemple point de rendez-vous)
Fin Tant Que
```

FIG. 4.22 – Algorithme du MigrationStrategyManager

4.5 Bilan

Pour résumer ce chapitre, nous avons étendu le système de migration de ProActive pour prendre en compte de nouvelles `Destinations` afin d'intégrer

l'administration réseau. Pour ce faire, nous avons : défini des destinations pour prendre en compte le modèle Client/Serveur en SNMP ; conçu une solution qui fournit les listes d'éléments constituant les itinéraires d'administration système et réseau par le biais d'`ItineraryServers` ; défini des types généraux d'itinéraires (les nœuds du réseau local, les équipements actifs du réseau local, etc..) en sous classant la classe de l'`ItineraryManager`.

Les `Destinations` que nous avons ajoutées au modèle de ProActive permettent d'obtenir des itinéraires d'administration système et réseau qui prennent en compte les équipements actifs du réseau, et le fait qu'un réseau d'entreprise puisse contenir plusieurs sous-réseaux, chacun de ceux-ci étant géré par un `ItineraryServer`. L'intégration de ces nouvelles destinations et leurs comportements sont récapitulés dans la table 4.6.

Destination	Action
NodeDestination	Migration et exécution de l'appel d'une méthode
SNMPDestination	Appel de méthode pour un accès en Client/Serveur SNMP
ISDestination	Appel de méthode permettant de contacter un ItineraryServer
CloneDestination	Appel de méthode permettant de créer des agents mobiles secondaires
RendezVousDestination	Appel de méthode permettant de synchroniser, les agents mobiles secondaires avec l'agent mobile père

TAB. 4.6 – Actions menées selon les types de Destination, version étendue

Le schéma général pour la construction d'un itinéraire selon le modèle que nous proposons (cf. figure 4.23) est le suivant :

A) Préparation de l'itinéraire (`ItineraryManager`) :

1. récupérer la référence de l'`ItineraryServer` (qui est présent en tant que service Jini)
2. récupérer la liste des éléments contenus dans l'`ItineraryServer`
3. transmettre pas à pas les éléments de l'itinéraire au `MigrationStrategyManager`

B) Faire suivre à l'agent mobile son itinéraire de manière automatique (via le `MigrationStrategyManager`)

– si une `ISDestination` est rencontrée alors le processus reprend :

1. préparer la suite de l'itinéraire (`ItineraryManager` A))
2. repartir en B)

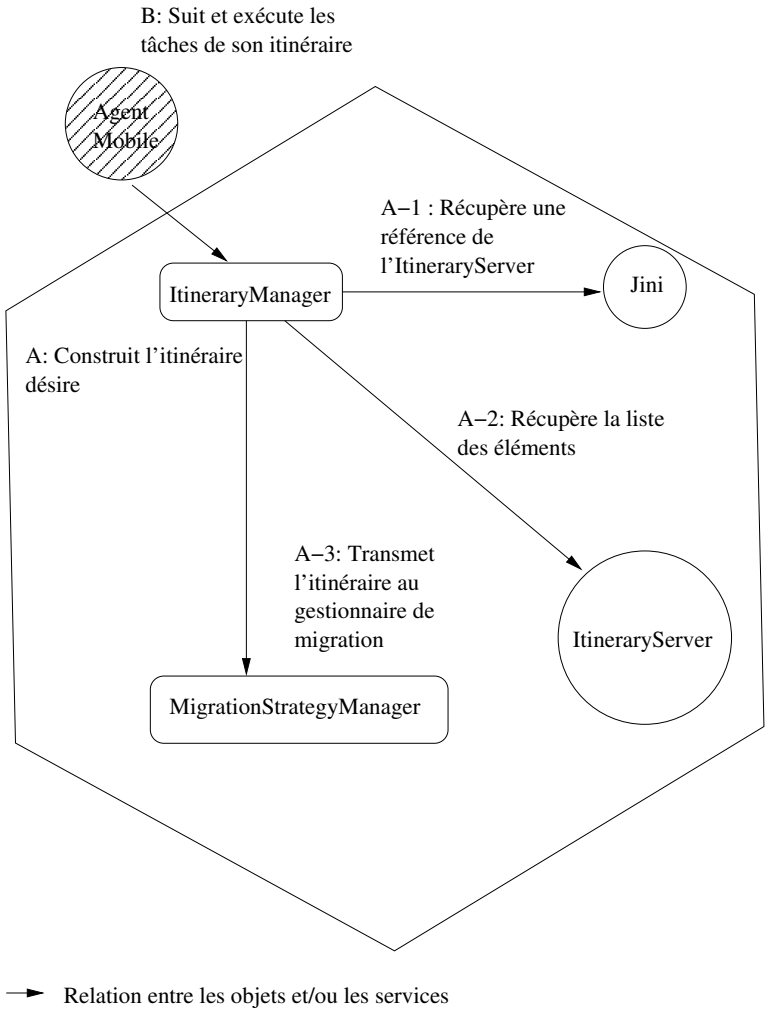

FIG. 4.23 – Fonctionnement de la création et du suivi d'un itinéraire

Chapitre 5

Conception et implantation d'une plate-forme d'administration système et réseau

5.1 Architecture générale : principe de conception

Dans l'objectif de concevoir une plate-forme à agents mobiles, nous avons cherché à distinguer les différents modules (ou services) qui composent cette plate-forme. Tout d'abord, il faut pouvoir connaître les éléments constituant le réseau pour fournir des informations utiles à un administrateur pour qu'il puisse effectuer son travail. Pour connaître ce réseau, il faut collecter et analyser les informations disponibles sur les éléments qui y sont connectés, et ce en découvrant les éléments du réseau. Ces éléments sont de type PCs, serveurs, routeurs, commutateurs, concentrateurs intelligents, imprimante en réseau, etc.., tout ce qui est en activité sur le réseau.

Cette plate-forme doit pouvoir mettre à disposition les informations collectées parce qu'elles décrivent le réseau, les éléments qui y sont connectés et qu'elles permettent de dialoguer avec les équipements actifs du réseau. De plus, ces informations permettront aux agents mobiles de se mouvoir sur le réseau à administrer.

La visualisation des données collectées du réseau sera présentée à l'administrateur par le biais d'une interface bâtie au dessus d'un module de supervision du cycle de vie des objets qui mettent en œuvre la plate-forme.

Le service de supervision offert par la plate-forme permettra aussi de piloter les agents mobiles d'administration, c'est-à-dire de les créer, de les interroger, de les déplacer, et de les arrêter, en somme d'intéragir avec eux.

5.2 Les services de la plate-forme

Nous allons présenter les services que nous avons pu déterminer comme étant indispensables dans une plate-forme à agents mobiles destinée à effectuer des tâches d'administration système et réseau. Cette présentation est décomposée en trois parties distinctes : le service de collecte, le service de mise à disposition de l'information et pour finir les services utilisés pour superviser le réseau et les agents mobiles d'administration.

5.2.1 Service de collecte et d'analyse

5.2.1.1 Service de description du réseau

Comme toute plate-forme d'administration système et réseau, il est nécessaire d'avoir, avant le démarrage du processus de collecte, des informations propres au réseau, c'est-à-dire un service de description du réseau, structuré par sous-réseaux. La description doit conserver cette structuration en sous-réseaux, parce que nous travaillons dans un contexte d'agents mobiles avec des itinéraires devant refléter la structure du réseau en sous-réseaux. Ces informations de description sont des informations naturellement connues statiquement donc il n'y a pas de contrainte pour offrir ce service de description. Il faut juste décrire ces informations une fois pour toute et les enregistrer dans un endroit ad hoc.

5.2.1.2 Service de collecte

En constatant que sur un réseau donné il y a une forte évolution des éléments connectés (ajout, suppression de matériels, par exemple), les informations utiles ne peuvent pas être données uniquement de manière statique, comme explicité ci-dessus, mais une partie de ces informations doit être obtenue dynamiquement. Par exemple, il est courant de nos jours de configurer le fonctionnement des éléments du réseau en utilisant le protocole DHCP (*Dunamic Host Configuration Protocol*[25]). Dans ce cas là, il est impossible de définir statiquement les éléments composant le réseau parce qu'une partie d'entre eux peut changer d'adresse IP. Ainsi les services offerts par ces éléments ne seront plus accessibles de la même manière. Il faudra donc les retrouver pour permettre l'accès aux services qu'ils offrent sur le réseau.

Remarque : Les services de description et de collecte sont des services complémentaires, car le service de collecte ne fonctionne que si le service de description du réseau définit l'environnement du réseau. Il est possible de décrire avec plus ou moins de précision le réseau à analyser : plus ce sera précis plus le temps d'exécution du service de collecte sera faible ; inversement, une description succincte du réseau entraînera un temps d'exécution de la collecte plus long.

5.2.2 Service de mise à disposition des informations collectées et analysées

Par rapport aux services décrits précédemment, le service de mise à disposition de l'information collectée fournira la liste des éléments découverts pendant la phase de collecte. Ce même service fournira la topologie du réseau découvert, structurée par sous-réseaux à administrer, représentant l'interconnexion des éléments découverts.

5.2.3 Service de supervision de la plate-forme

Il s'agit du service qui permet la visualisation, l'utilisation et la supervision des objets composant la plate-forme.

Nous trouvons le service de pilotage, pour l'ensemble du réseau, des objets de la plate-forme, objets mettant en œuvre les deux grandes familles de services que nous venons de décrire ci-dessus. Ce service de supervision pourra collecter les informations décrivant le réseau, d'une part à des fins de sauvegarde sur la station d'administration, et d'autre part pour avoir une vue en temps réel des éléments découverts automatiquement et de leur interconnexion.

La supervision des agents mobiles sera effectuée par un service de pilotage permettant de localiser automatiquement les agents mobiles pendant leur tâche d'administration par le biais d'un outil tel `IC2D` [7] (livré en standard avec ProActive) et d'un logiciel *ad hoc* à la plate-forme permettant entre autre, d'interroger à distance des agents mobiles d'adminstration.

5.3 Architecture générale : mise en œuvre

5.3.1 Introduction

L'architecture de notre plate-forme se compose de trois grandes familles de services : les services de description du réseau et les services de scrutation et de collecte de l'information du réseau ; les services mettant à disposition l'information collectée, par exemple la liste des éléments détectés, mais aussi la topologie du réseau construite ; la dernière famille de service comprend quant à elle tous les services de supervision qui permettent d'avoir l'état général du réseau et aussi des outils permettant d'exploiter cette plate-forme à agents mobiles dans un cadre d'administration système et réseau.

Pré-requis : Afin d'utiliser de manière complètement transparente cette plate-forme à agents mobiles, un service d'enregistrement Jini [53] doit être installé et accessible, afin que les services puissent être mis à la disposition du domaine

d'administration. Plus techniquement, cela impose le passage du trafic multicast entre les différents sous-réseaux de l'entreprise utilisant une telle plate-forme.

5.3.2 Connaissance du réseau

Nous allons décrire les services qui permettent d'obtenir la connaissance des éléments connectés sur le réseau et la topologie du réseau obtenue par corrélation des informations collectées.

5.3.2.1 Service de description d'un sous-réseau

Cette information doit exister avant le déploiement de la plate-forme, au même titre que statiquement dans des fichiers ou dans une base de données, comme dans MobileSpaces [78], Hp Openview [35] ou Nagios [57]. Les trois informations minimales afin de pouvoir collecter de l'information sur un sous-réseau sont : l'adresse du routeur par défaut (appelé aussi *default gateway*), le masque du réseau (par exemple 255.255.255.0 pour une classe C d'adresse IP 192.168.10.0) et le nom de la communauté SNMP en lecture (par défaut `public`) pour accéder aux informations de la MIB des agents SNMP. Au contraire des autres plates-formes, nous voulons rendre disponible cette information en s'abstrayant d'une localisation fixe, statique. C'est pour cela que cette information est encapsulée dans un objet Java dont on enregistre une copie dans un ou plusieurs Lookup Services de Jini. Il suffira de récupérer une copie à la demande, par le biais d'une opération de recherche auprès d'un des Lookup Service (pour de plus amples explications de notre utilisation de Jini, voir l'annexe en section 9.1).

Plus concrètement, chaque sous-réseau est décrit par le biais des informations présentées par la table 5.1 :

De cette table, remarquons que nous avons prévu de prendre en compte certains éléments, en plus des informations générales décrivant le sous-réseau, afin de pouvoir spécifier certaines valeurs particulières :

- `SNMPDestination` : définir un élément permanent ayant un agent SNMP dont les noms de communautés SNMP n'ont pas les même valeurs que les valeurs par défaut, ou un élément composant le cœur du sous-réseau (par exemple un commutateur, serveur DNS, etc...)
- `NodeDestination` : définir un nœud de la plate-forme ProActive actif en permanence. Ce nœud peut être hébergé, par exemple, sur un serveur ayant des capacités de traitement suffisantes pour l'administration réseau

Ces informations peuvent être saisies par le biais de l'onglet graphique présenté par la figure 5.2, à partir desquelles un objet décrivant un sous-réseau est instancié en vue d'être enregistré. Les informations saisies dans cet onglet sont les mêmes que celles présentées par la table 5.1. Une interface de saisie générique a été définie pour saisir les `Destinations` permanentes du sous-réseau. Par

Nom du TAG	Description	Exemple de Valeur
NetworkName	Nom logique pour le sous-réseau	reseau-168-80-jini
NetworkAddress	Adresse IP du sous-réseau	192.168.80.0
NetworkMaskAddress	Masque de sous-réseau	255.255.255.0
NetworkBroadcastAddress	Adresse IP de broadcast du sous-réseau	192.168.80.255
NetworkRouter	L'adresse IP du routeur par défaut	192.168.80.253
SnmpReadDefaultCommunity	Commutauté SNMP par défaut en lecture	public
SnmpReadWriteCommunity	Communauté SNMP par défaut en lecture/ecriture	private
`Destination`	une Destination selon le modèle de nos destinations	

TAB. 5.1 – Explication des Tags du service de description d'un sous-réseau

exemple la figure 5.3 permet de visualiser tous les constructeurs possibles permettant de créer une `SNMPDestination`. On montre aussi la `Javadoc` associée afin que l'utilisateur prenne connaissance de la sémantique (informelle) des différents paramètres des constructeurs. Une fois le constructeur sélectionné, la figure 5.4 montre l'onglet de saisie associé.

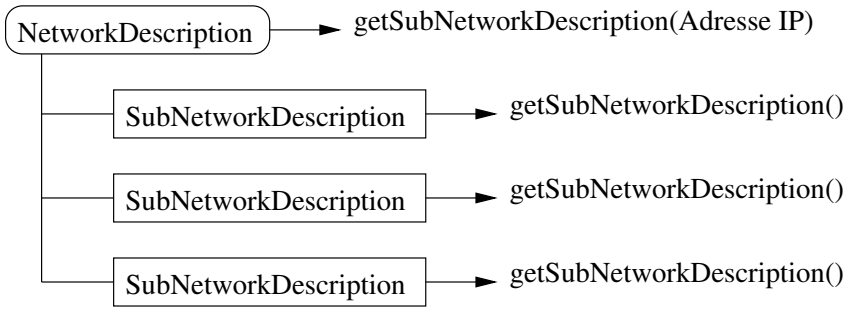

FIG. 5.1 – Hiérarchie des descriptifs des sous-réseaux

La méthode de l'objet, appelé `SubNetworkDescription`, représentant ce service est : `getSubNetworkDescription()` pour obtenir le descriptif de tout le sous-réseau. Un autre objet appelé `NetworkDescription` qui regroupe les objets de type `SubNetworkDescription`, et offre une méthode `getSubNetwork-Description(Adresse IP)` pour obtenir le descriptif d'un sous-réseau particulier (cf. figure 5.1).

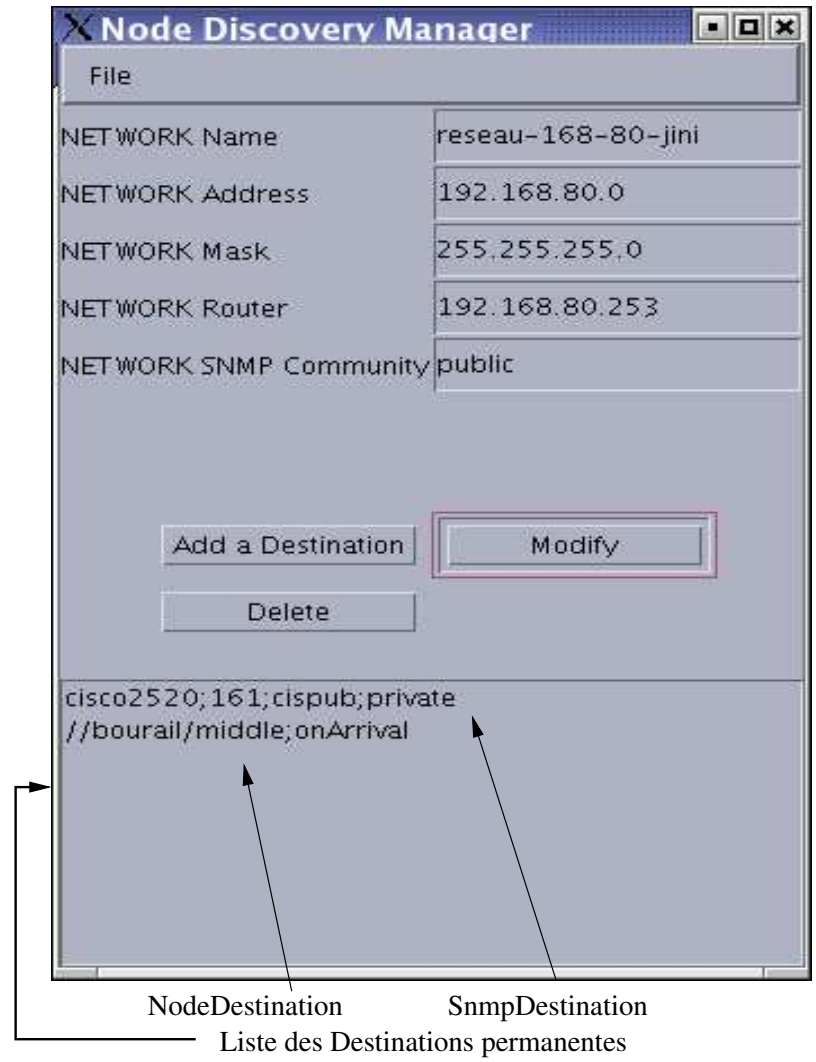

FIG. 5.2 – Interface pour la saisie du descriptif d'un sous-réseau

5.3.2.2 Service de découverte d'un sous-réseau

Un tel service requiert une activité propre afin d'effectuer la découverte d'un sous-réseau, à intervalles de temps réguliers. Un tel besoin justifie l'utilisation de la notion d'object actif, interrogeable afin de récupérer les éléments découverts [72]. L'activité d'un tel objet consiste en l'exploration du sous-réseau (voir la section 5.4 pour l'algorithme mis en œuvre). Le résultat de cette activité est composé de deux structures de données :

- la liste des éléments du sous-réseau (a fortiori la liste des éléments SNMP et la liste des nœuds de la plate-forme ProActive)
- la topologie détectée, présentée sous forme arborescente

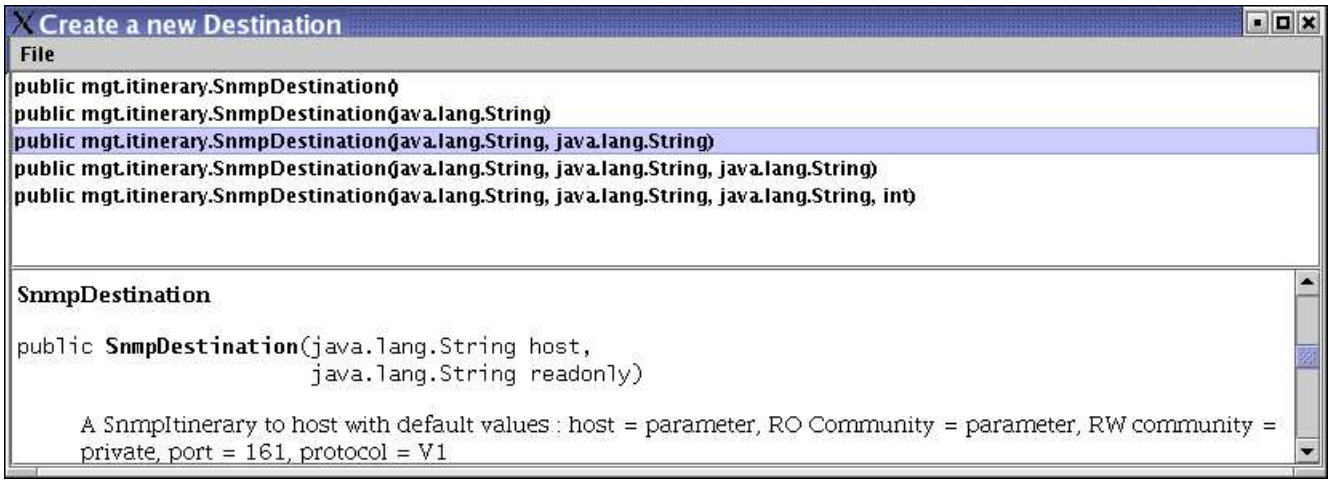

FIG. 5.3 – Interface pour la sélection du constructeur de la `Destination`

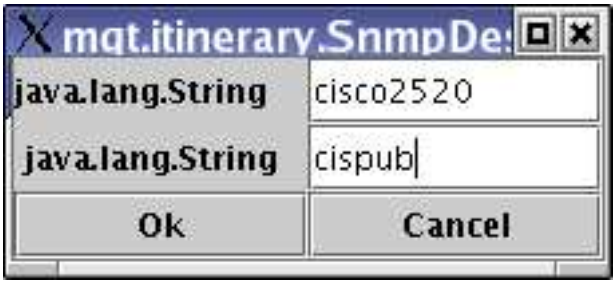

FIG. 5.4 – Interface pour la saisie d'une `Destination`

On alimente dès que possible le service de mise à disposition parce que l'on veut avoir les données toujours disponibles.

Les méthodes publiques de cet objet sont :
`startTopologyDiscovery(SubNetworkDescription)` pour lancer le processus de collecte et d'analyse du sous-réseau et `stopTopologyDiscovery()` pour arrêter si nécessaire ce processus. Les services offerts par cet objet aux autres objets de la plate-forme sont : `getSNMPElements()` pour obtenir la liste des éléments SNMP du sous-réseau, `getProActiveNodes()` pour obtenir la liste des nœuds de la plate-forme ProActive en activité et `getTopologyMap()` pour obtenir l'arbre représentant la topologie du sous-réseau.

5.3.3 Service de mise à disposition des informations découvertes

Un administrateur sait que le processus de découverte des éléments d'un sous-réseau n'est pas une action qui s'exécute instantanément. Ainsi, la lenteur relative du processus de découverte (quelques dizaines de secondes pour quelques dizaines

d'éléments) justifie la présence d'un cache des informations découvertes. Lors de l'invocation du service de mise à disposition, on renvoie les informations présentes dans le cache. Les méthodes associées sont : `getSNMPElements()` pour la liste des éléments SNMP, `getProActiveNodes()` pour la liste des nœuds ProActive et `getTopologyMap()` pour récupérer la topologie construite. Quant au cache, il est mis à jour automatiquement grâce à ces méthodes : `setTopologyMap()`, `setSNMPElements()` et `setProActiveNodes()` par le service de découverte du réseau.

Pour des raisons à la fois d'exactitude et de rapidité de mise à disposition, le cache est mis à jour au fur et à mesure qu'un nouvel élément est découvert, et sera complètement actualisé en fin du processus de découverte, lorsque cette fin est signalée.

Pour que le service soit accessible, notamment pour construire des itinéraires, mais aussi pour visualiser les informations mises à disposition pour un sous-réseau donné, le service va s'enregistrer automatiquement auprès d'un Lookup Service.

5.3.4 Service de supervision

Il s'agit de piloter la création des services de mise à disposition, services qui eux même font le nécessaire pour déclencher le service de découverte. Un service de mise à disposition est créé en fonction de chaque description de sous-réseau. Si sur un sous-réseau donné existe un nœud permanent, le service peut être créé directement sur le sous-réseau, plutôt que sur le nœud courant. Le fait que le service de mise à disposition s'exécute sur un nœud permanent présente l'avantage que ce service sera toujours disponible.

Le service de supervision utilise le service de mise à disposition de chacun des sous-réseaux décrits (via le service de description du réseau) afin de collecter une vue globale du réseau à administrer, à l'aide des différentes vues partielles pour chacun des sous-réseaux. Les méthodes définies par ce service sont :
`createItineraryService(SubNetworkDescription)` pour créer le service de mise à disposition, `getGlobalTopology()` pour récupérer le schéma de la topologie globale du réseau, `getGlobalSNMPElements()`,
`getGlobalProActiveNodes()` pour récupérer les listes des éléments du réseau (SNMP et nœuds de ProActive).

Une fois les objets de la plate-forme déployés, leur pilotage peut être effectué via l'outil générique de supervision de la plate-forme ProActive, `IC2D` [7] (cf. figure 5.5). Par exemple, il est possible utiliser IC2D pour changer l'emplacement d'objets actifs implantant les différents services afin de rééquilibrer la charge CPU (cf. projet D-Raje [32] pour obtenir de manière portable la charge CPU) consommée sur un seul serveur (cf. figure 5.6). Grâce au mécanisme de *forwarding* de ProActive (suivi des messages vers la nouvelle localisation de l'objet actif), ces objets restent joignables. Si de plus ils sont enregistrés dans Jini, lors de l'expiration du bail, ils se réenregistreront en indiquant à ce moment là leur

localisation courante.

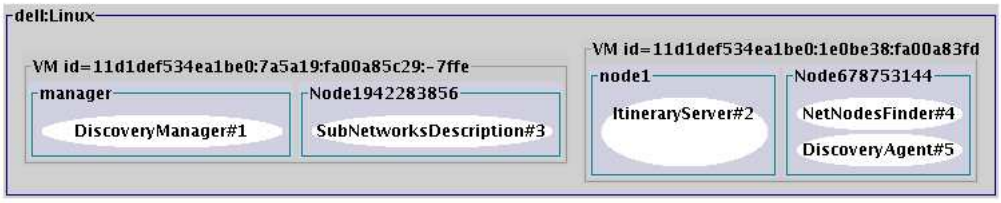

FIG. 5.5 – Les services de la plate-forme vus par IC2D

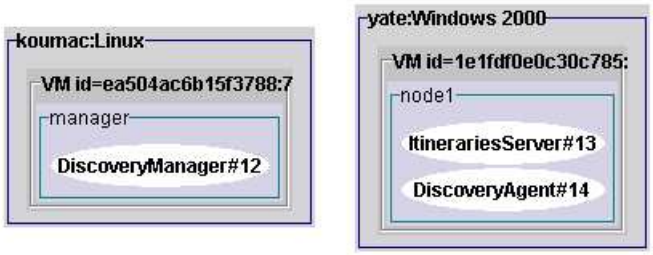

FIG. 5.6 – Répartition des services de la plate-forme sur deux nœuds différents

5.3.5 Récapitulatif

Les différents services de notre plate-forme sont mis en œuvre par des objets Java distincts, qui sont présentés par la figure 5.7.

Le service de supervision est le service permettant de présenter et de surveiller tous les services de la plate-forme : service de description du réseau, service de découverte d'un sous-réseau et le service de mise à disposition des informations découvertes. Il est implanté par l'objet actif `DiscoveryManager`.

Le service de description de réseau est un service qui s'enregistre dans un Lookup Service Jini, implanté par un objet (non actif) appelé le `Network-Description`, comme présenté dans la section 5.3.2.1.

Le service de découverte d'un sous-réseau est un service qui est implanté par deux objets actifs de ProActive : le `DiscoveryAgent` qui est l'agent de découverte des éléments du réseau et de la topologie ; le `NetNodesFinder` qui est un agent de localisation des nœuds ProActive qui sont en cours d'exécution. La mise en œuvre de ces deux agents distincts est faite afin d'obtenir un délai d'exécution de chacun de ces services plus rapide.

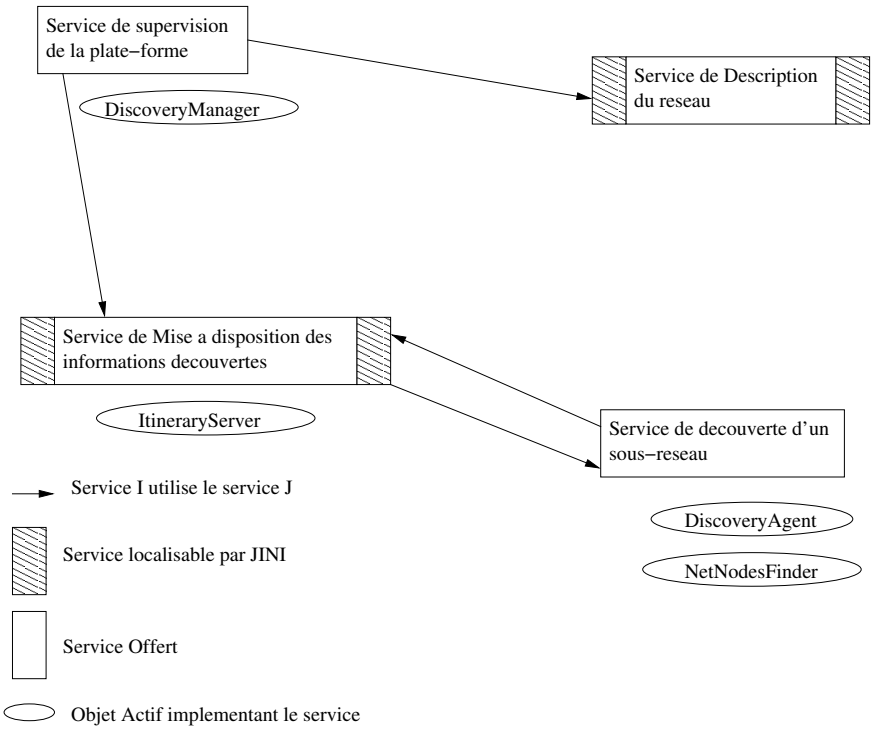

FIG. 5.7 – Le récapitulatif des services de notre plate-forme à agents mobiles pour l'administration système et réseau

Le service de mise à disposition des informations découvertes est le service qui sert de cache de données au service de découverte d'une part, et d'autre part qui permet d'être interrogé par les agents mobiles d'administration lors de la construction d'un itinéraire d'administration et par le ou les `DiscoveryManagers`. Ce service est un service qui s'enregistre dans un Lookup Service Jini afin d'être accessible indépendamment de sa localisation.

5.3.6 Interface graphique

Il est bien évident que les administrateurs requièrent d'avoir une interface graphique (GUI) de supervision pour la plate-forme d'administration système et réseau. On va évidemment faire appel aux fonctionnalités du service de supervision de la plate-forme qui sait faire remonter l'ensemble des informations pertinentes pour l'aspect graphique. A chaque GUI est donc associé une instance du service de supervision de la plate-forme, c'est-à-dire ce que nous avons appelé un `DiscoveryManager`.

5.3.6.1 Les informations du réseau

Ainsi nous proposons une vue graphique et globale du réseau à administrer grâce aux informations que nous avons collectées sur le réseau (cf. figure 5.8).

La figure 5.9 présente l'onglet de visualisation des informations qui ont été collectées sur un élément du réseau. On y retrouve le nom logique de l'élément (nom DNS), son adresse IP et l'adresse Ethernet qui a été découverte.

5.3.6.2 Pilotage et interrogation d'agents mobiles

De plus, cette GUI va contribuer au service de pilotage des agents mobiles d'administration, en plus d'IC2D qui permet de visualiser, voire de migrer manuellement des agents (cf. figure 5.10).

On va pouvoir graphiquement préparer et déclencher la création d'agents mobiles accompagnés d'un itinéraire, puis déclencher des invocations de services sur un agent mobile en activité. Notre outil ad hoc nous permettant de démarrer des agents mobiles pour leur faire suivre un itinéraire pré-défini est présenté par la figure 5.11, par la sélection de la classe de l'agent mobile et du type d'itinéraire que l'on souhaite lui faire suivre. La figure 5.12 présente l'interrogation de cet agent mobile à distance. Dans l'exemple présenté, l'agent mobile donne des informations sur l'état du système sur lequel il s'est positionné en attendant un nouvel ordre de migration.

5.3.6.3 Supervision des ressources d'un élément

Nous avons doté notre bibliothèque d'outils permettant la supervision des éléments du réseau, et ce directement à partir de l'interface graphique ou bien à partir de notre outil ad hoc destiné à contrôler le fonctionnement des agents mobiles d'administration (cf. figure 5.11). La figure 5.13 présente le résultat obtenu par un agent mobile effectuant la supervision d'un élément du réseau. Il s'agit d'obtenir, quel que soit le système Linux ou Windows, la charge CPU et l'espace de mémoire libre sur le système.

Pour avoir la charge réseau, nous utilisons un onglet de la GUI d'administration (cf. figure 5.14) qui permet de collecter les informations de trafic sur l'élément cible. On obtient une collecte des informations de l'élément toutes les 5 secondes afin d'avoir un aperçu rapide de la charge de l'élément.

5.3.6.4 Plate-forme multi-administrateurs

Comme une GUI fait appel à sa propre instance du service de supervision de la plate-forme (l'instance du `DiscoveryManager` qui lui est associée), il est tout à fait envisageable d'avoir plusieurs GUI actives en même temps. L'intérêt évident est de permettre d'avoir soit plusieurs administrateurs du même réseau

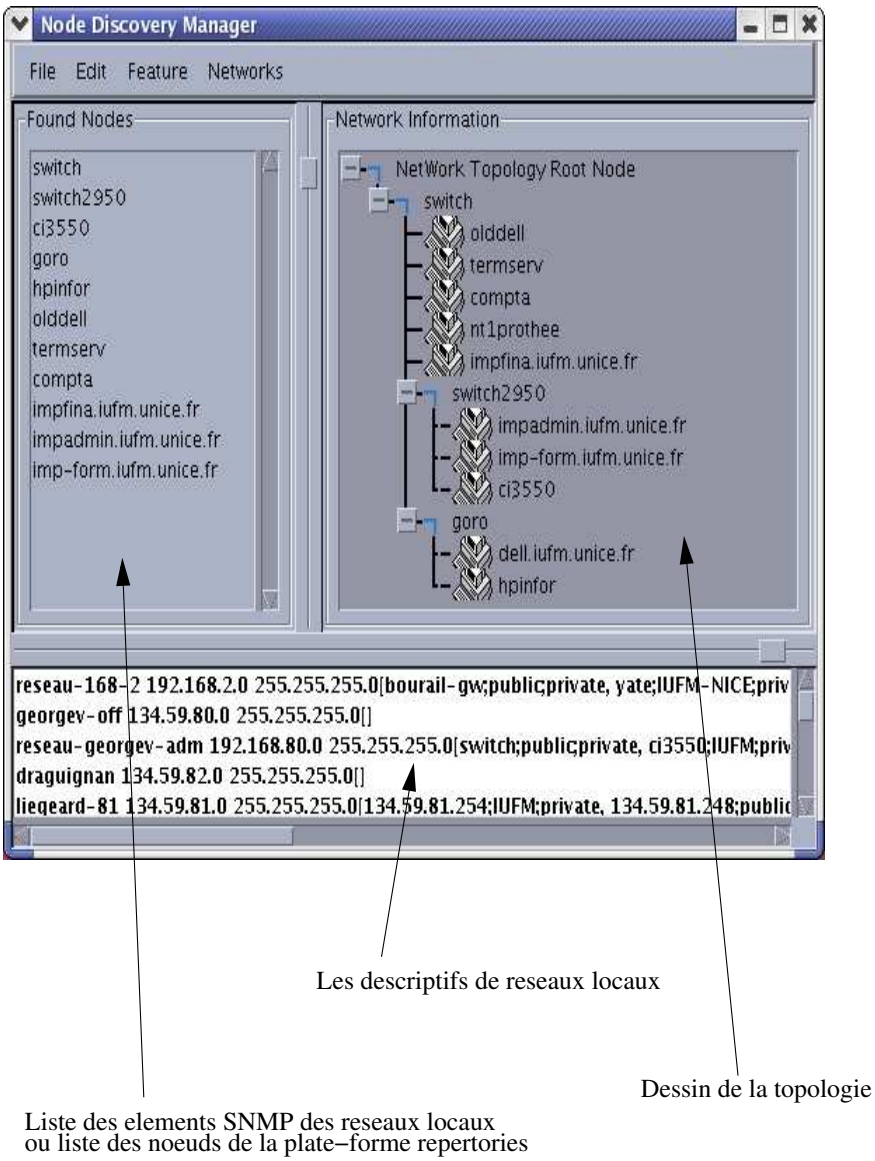

FIG. 5.8 – Interface de visualisation de la topologie

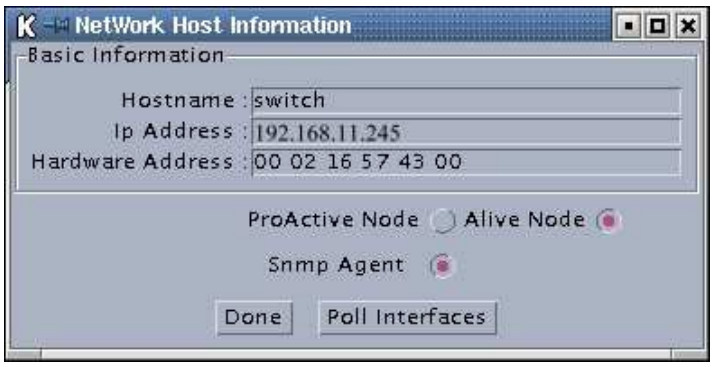

FIG. 5.9 – Onglet de visualisation des informations d'un élément

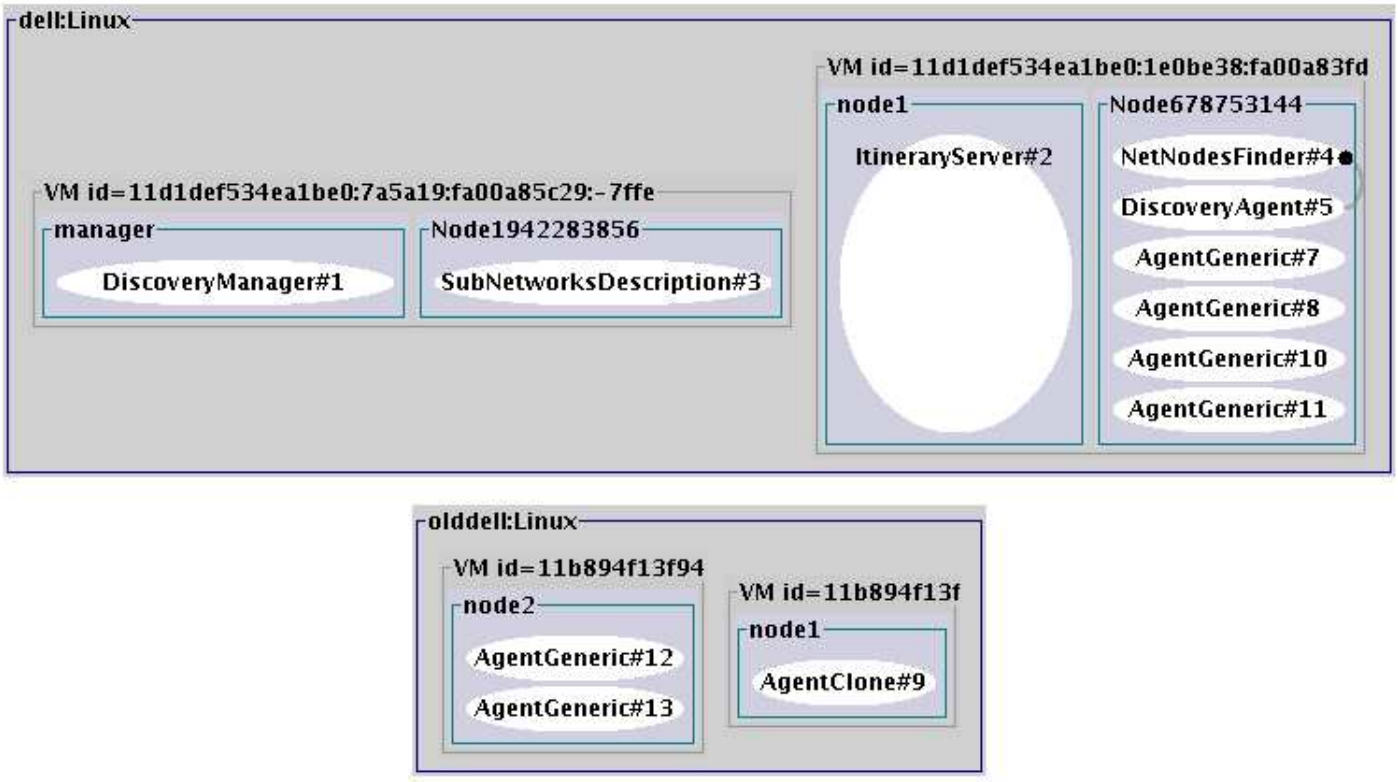

FIG. 5.10 – Les services de la plate-forme et des agents mobiles d'administration vus par IC2D

en activité simultanément, soit un administrateur mobile qui a démarré autant de services de supervision que d'emplacements physiques différents où il passe.

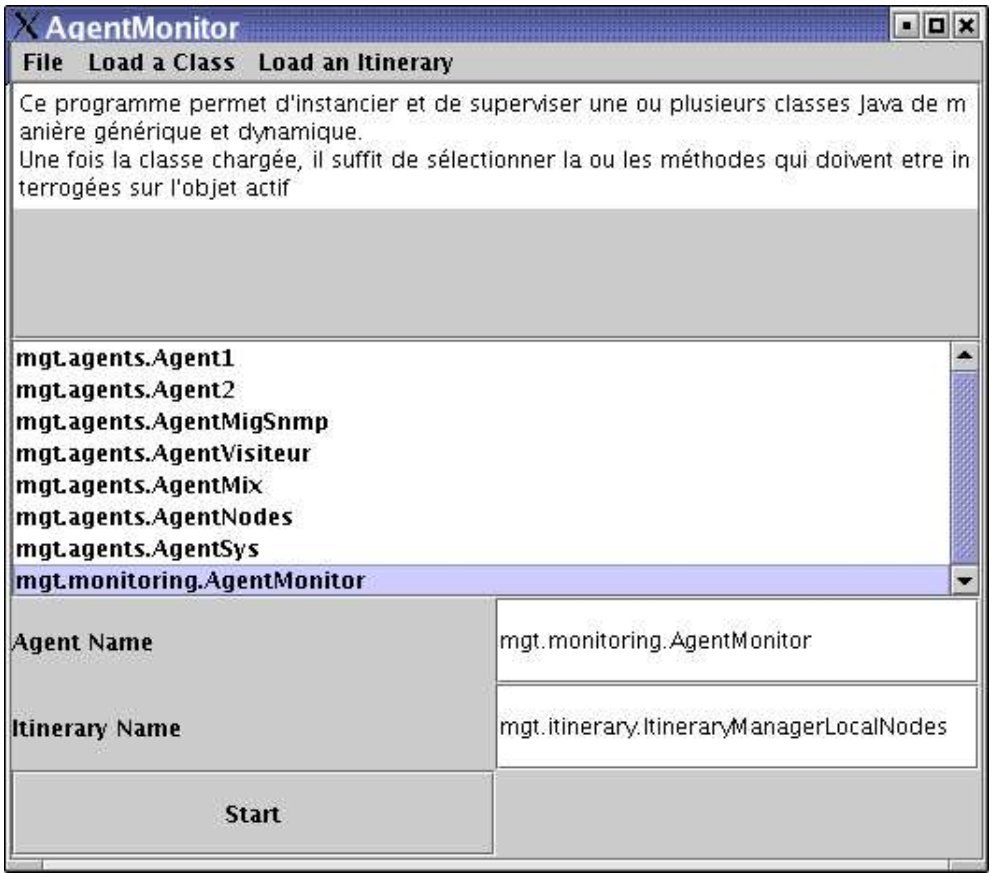

FIG. 5.11 – Lancement d'un agent mobile d'administration

5.4 Principes et implantation de l'algorithme de découverte de la topologie

5.4.1 Introduction

Dans un réseau donné, c'est-à-dire défini par une adresse de réseau unique, nous allons construire la topologie de niveau 2 du réseau. Nous allons déterminer l'interconnexion des éléments dans le réseau et nous n'allons pas déterminer la topologie physique car l'obtention d'une telle topologie est impossible techniquement (il s'agit du câblage à proprement parler). Notre topologie de réseau sera décrite comme un arbre n-aire, dont la racine est l'élément actif avec une adresse IP (possédant un agent SNMP) tête du réseau, c'est-à-dire l'élément qui a la charge la plus élevée, car effectivement tous les liens physiques qui mènent vers les différentes branches du réseau sont connectés à lui. Nous utiliserons donc le service de description du réseau (`NetworkDescription`) pour obtenir l'adresse IP du réseau pour lequel on désire construire la topologie.

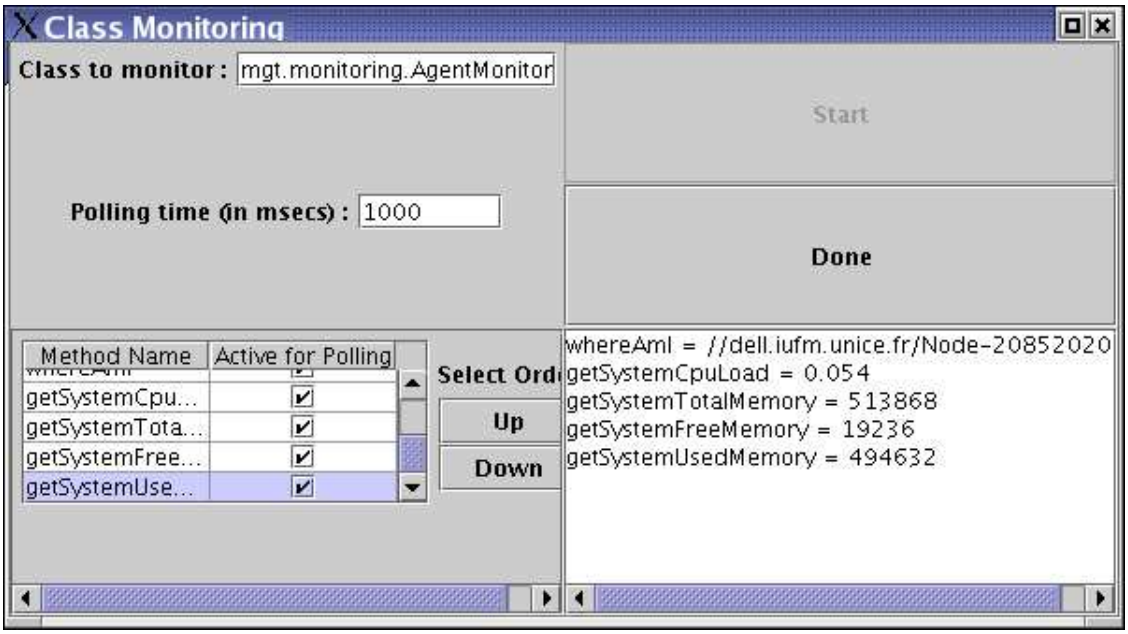

FIG. 5.12 – Interrogation à distance de l'agent mobile via la GUI

FIG. 5.13 – AgentMonitoringFrame : Supervision des ressources système d'un élément

5.4.2 Phase de détection des éléments

Étant donné un `SubNetworkDescription` (cf. section 5.3.2.1), nous allons détecter tous les éléments qui ont la même adresse de réseau (label `Network Address`) et qui génèrent du trafic. Nous allons lire la table ARP du routeur par défaut du réseau (indiqué par le label `Network Router` dans la table 5.1). A partir des informations collectées, on va construire une table ARP globale de façon incrémentale. La détection d'un nouvel élément, c'est l'occurrence d'un élément dans une entrée d'une table dont l'adresse IP était inconnue jusqu'alors. Récur-

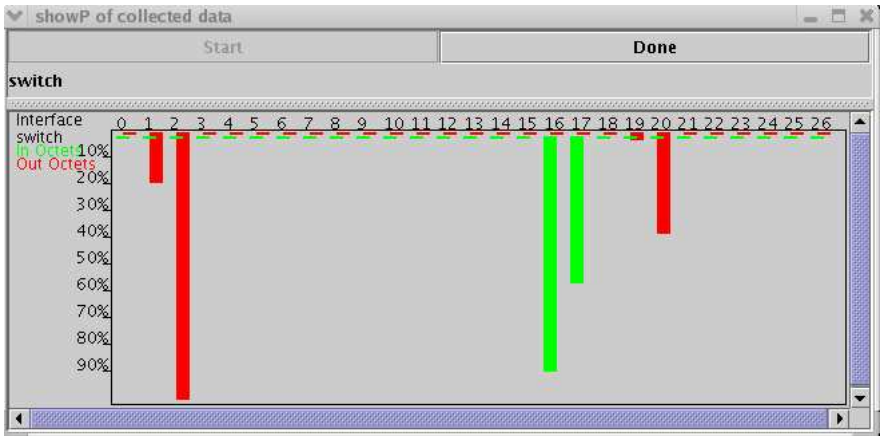

FIG. 5.14 – Onglet de visualisation de la charge réseau d'un équipement actif

sivement, la table ARP de chaque nouvel élément sera fusionnée à la table ARP globale en cours de construction. Cela suppose qu'on y a accès, ce qui est le cas s'il y a un agent SNMP sur l'élément en cours. De manière naturelle, l'algorithme se termine lorsque tous les éléments ainsi découverts ont déjà été explorés. Pour forcer les éléments, même générant peu de trafic[1], à en engendrer, on peut au préalable générer un paquet de type `ICMP`, sur l'adresse de `broadcast` du réseau (par la commande ping, cf. section 2.1). Chaque élément va alors répondre à l'émetteur.

Informations collectées : Pour chaque élément (avec un agent SNMP) répertorié ou détecté, on collecte les informations suivantes :

- les services fournis : agent SNMP, nœud de ProActive
- le type de service réseau fourni (System)
- la table de routage (ip.ipRouteTable)
- les services IP fournis (tcp.TcpConnTable et udp.UdpTable)
- En fonction du type de service réseau fourni (system.sysServices) :
  - la matrice de commutation (dot1dBridge.dot1dTp.dot1dTpFdbTable) (dans le cas d'un commutateur)
  - les informations du Spanning Tree Protocol (dot1dBridge.dot1dStp.dot1dStpPortTable) (dans le cas d'un commutateur)
  - les informations de gestion du répéteur (snmpDot3RptrMgt.rptrAddrTrackPortInfo) (dans le cas d'un concentrateur Intelligent)

[1]Le minimum de trafic généré par un élément en réseau correspond à l'échange d'un paquet de vérification de l'unicité de l'adresse IP qui lui est allouée par l'administrateur (protocole ARP)

Remarque 1 : la collecte des informations sur la présence d'un nœud ProActive peut se faire de manière indépendante, et donc en parallèle de l'algorithme itératif de détection, ce qui économise environ 1/3 du temps de détection total.

Remarque 2 : la récupération de l'information de présence d'un agent SNMP peut être très performante si on diffuse un paquet SNMP v1 [2] (le protocole SNMP V1 est encore utilisé pour la collecte des données [22]) , auquel répondront les agents SNMP qui seront ainsi mis, au plus tôt, dans la liste des éléments découverts.

5.4.3 Phase de connexion des éléments

Ce n'est pas tout de produire la liste des éléments découverts sur le réseau, on veut aussi détecter de quelle manière ils sont reliés entre eux, pour connaître comment passe le trafic réseau entre eux. Les principes, et quelques détails techniques lorsque nécessaire, sont donnés dans cette section.

Étant donnée une adresse Ethernet, c'est-à-dire correspondant à un élément ayant aussi une adresse IP, on cherche : si cette adresse apparaît dans une entrée d'une table ARP d'un autre élément détecté, on en déduit le numéro de l'interface par laquelle l'adresse Ethernet a été apprise (l'interface d'entrée) ; s'il s'agit de rechercher cette adresse dans les informations collectées dans l'agent SNMP d'un commutateur, alors on utilisera aussi la matrice de commutation pour obtenir le numéro de l'interface correspondante. Voici ci-dessous une explication plus détaillée de l'usage que nous faisons de la matrice de commutation.

La matrice de commutation La matrice de commutation d'un commutateur (*forwarding database*) permet à un tel équipement actif de rediriger le trafic entrant vers la bonne interface de sortie. Cette matrice de commutation donne les associations entre les adresses Ethernet des éléments du réseau local et les interfaces du commutateur. Par exemple le numéro d'interface 38 sur l'exemple présenté par la table 5.2, correspond à l'adresse Ethernet 00 :04 :76 :97 :86 :46. C'est-à-dire que si un paquet est pour cette adresse Ethernet, il doit être retransmis par le numéro d'interface 38 de notre exemple.

Après de telles déductions, en considérant tous les éléments découverts, on obtient une liste exhaustive qui correspond en fait à l'énumération de tous les chemins de longueur supérieure ou égale à 1 par lesquels du trafic a transité. Par exemple, supposons le petit réseau sur la figure 5.15 par le biais duquel le PC imprime.

[2] avec l'adresse de diffusion du réseau (label `NetworkBroadcastAddress`), le nom de la communauté SNMP définit par défaut (label `SnmpReadDefaultCommunity`) et l'*oid* correspondant à la variable *system.sysDescr*

Forwarding Database d'un commutateur		
Index de hashage	dot1dTpFdbAddress Adresse Ethernet	dot1dTpFdbPort Le port de sortie
0.4.118.151.134.70	00 :04 :76 :97 :86 :46	38
0.4.118.163.61.144	00 :10 :7B :3A :20 :B5	22
1.128.194.0.0.0	01 :80 :C2 :00 :00 :00	0
1.128.194.0.0.1	01 :80 :C2 :00 :00 :01	0

TAB. 5.2 – Extrait de la MIB BRIDGE dot1dTpFdb

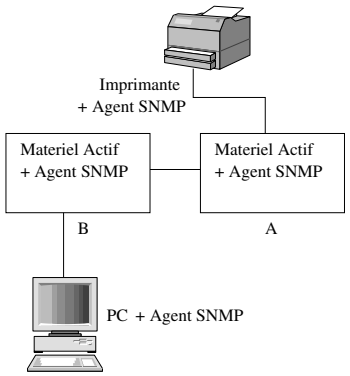

FIG. 5.15 – Schéma d'un réseau quelconque

L'étape des déductions fournit la liste de la table 5.3. On voit cependant sur cet exemple qu'il n'existe pas de lien direct entre le PC et l'imprimante (pourtant la liste obtenue laisse possible une telle éventualité).

Source → Destination			
B → PC	PC → B	A→B	Imprimante→A
B→A	PC→A	A→PC	Imprimante→B
B→Imprimante	PC→Imprimante	A→Imprimante	Imprimante→PC

TAB. 5.3 – Liste résultante de l'énumération des chemins

5.4.3.1 Détermination de la topologie

Il s'agit donc d'arriver à éliminer des éléments de cette liste de sorte à ne conserver que ceux qui correspondent à des connexions directes.

Repérer les liens qui ne peuvent exister par nature : les principes ci-dessous sont utilisés :

– étant donné 2 éléments autre que des commutateurs, routeurs ou concentrateurs, qui *semblent* être directement connectés : si on ne détecte aucun élément de type commutateur, routeur ou concentrateur entre eux, alors on peut en conclure qu'ils ne sont pas directement connectés.
– étant donné 2 éléments ne possédant chacun qu'une seule interface réseau (à l'exception d'un routeur), alors il n'est pas possible qu'il existe une connexion physique directe entre eux.

B→PC	PC→B	A→B	Imprimante→A
B→A	PC→A	A→PC	Imprimante→B
B→Imprimante		A→Imprimante	

Tab. 5.4 – Liste résultante de l'énumération des chemins sans connexion impossible

Reprenons l'exemple précédant : étant donnée la liste de la table 5.3, on obtient ainsi la liste expurgée des connexions directes impossibles présentée par la table 5.4.

Repérer les liens de connexions indirectes : On essaye de repérer, par transitivité, des chemins de longueur supérieure à 1 parmi les éléments restant dans la liste. Pour cela, l'idée est de détecter la présence d'au moins un élément de type commutateur (ou concentrateur intelligent). Le principe est le suivant :
– étant donné une paire Source-Destination (plus précisément Destination + numéro de l'interface d'entrée) : intuitivement, un élément intermédiaire est supposé exister car il existe en plus du chemin Source-Destination, il existe au moins un autre chemin qui aboutit à la destination sur la même interface (par exemple PC→B, PC→A et B→A).

Étant donné la liste de la table 5.4, on obtient ainsi la liste expurgée des connexions indirectes, présentée par la table 5.5.

B→PC	PC→B	A→B	Imprimante→A
B→A			
		A→Imprimante	

Tab. 5.5 – Liste résultante de l'énumération des chemins sans connexion impossible et indirecte

5.4.3.2 Orientation de la topologie

La topologie (sommets, liens bidirectionnels reliant ces sommets) est donc obtenue. On peut se la représenter comme un graphe. Néanmoins, une telle to-

pologie n'a un sens pour un administrateur que si elle est orientée selon le trafic (montré comme étant un arbre), c'est-à-dire que les nœuds du graphe obtenu sont des équipements actifs. Soit en fonction du trafic à destination de l'extérieur du réseau local (dans ce cas là, la racine de la topologie sera un routeur) ; soit en fonction du commutateur désigné comme le *Designated Root* conformément à l'algorithme du Spanning Tree. De ce fait, l'orientation de la topologie doit mener au fait que pour toute paire d'éléments (e1,e2) on a soit (e1→e2) ou sinon (e2→e1). Nous devons donc repérer les liens qui ne participent pas au trafic ascendant (des feuilles vers la racine). On ne sait pas déterminer à la seule observation des tables ARP, le sens du trafic (ascendant ou descendant). On est obligé de déduire le sens en fonction de la nature des éléments et donc des services qu'ils rendent au sein du réseau.

Détermination des feuilles : Typiquement, si un élément n'a qu'une seule interface physique, il est forcément une feuille et donc, on supprimera tout lien direct pour lequel cet élément serait indiqué comme étant la destination. Ou bien si un élément E n'a pas d'agent SNMP malgré que l'on puisse avoir détecté un lien de cet élément vers un autre élément, ou inversement de cet autre élément vers E, on décide de ne conserver que le lien de E vers cet autre élément en admettant que c'est parce qu'il participe au trafic ascendant. En l'absence d'informations SNMP plus complètes, on ne peut pas faire de déduction plus précise.

Détermination des éléments nœuds : Essentiellement, il s'agit de déterminer le sens dans lequel les éléments qui composent le cœur du réseau (commutateurs, concentrateurs intelligents, routeurs) sont interconnectés. Vu que la gestion du trafic entre commutateurs est déjà forcément orientée suite à l'application de l'algorithme du Spanning Tree, il suffit de consulter les informations qui en ont résulté. Ainsi on obtient assez simplement (plus de détail sont donnés dans le paragraphe "Description de l'algorithme du Spanning Tree" ci-dessous) la portion de topologie concernée.

Étant donné la liste de la table 5.5, on obtient ainsi la liste présentée par la table 5.6 permettant de construire un arbre représentant la topologie du réseau de racine `A`, en supposant que l'algorithme du Spanning Tree ait désigné le commutateur A comme *DesignatedRoot*.

B→A	PC→B	Imprimante→A

Tab. 5.6 – Liste résultante pour la construction de la topologie

A priori, la topologie est exacte et précise. On peut, par exemple, la visualiser graphiquement ou s'en servir en vue de construire des itinéraires (cf le chapitre

suivant). Notre algorithme appliqué sur le réseau présenté par la figure 5.15 a pour topologie résultante le schéma proposé par la figure 5.16.

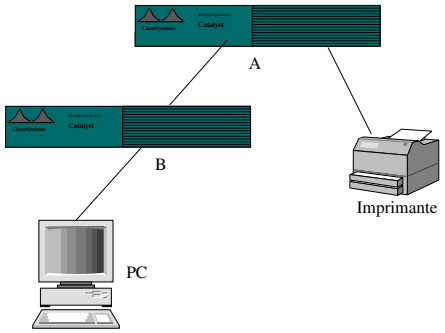

FIG. 5.16 – Topologie obtenue par l'algorithme

Néanmoins pour améliorer la représentation visuelle donnée à l'administrateur, on sait rajouter des éléments qui ne peuvent pas être détectés parce qu'ils n'ont pas d'adresse IP, bien qu'ils participent effectivement à la réémission de signaux réseau (tels que les concentrateurs). Un concentrateur qui ne disposerait pas agent SNMP, est détectable de manière implicite lors de la phase de connexion. En effet, si sur une interface d'un commutateur, on a plusieurs adresses Ethernet apprises, c'est qu'il y a un concentrateur physiquement relié à cette interface, et sur lequel sont connectés les éléments correspondant à ces adresses Ethernet.

Bien sûr, si un concentrateur disposait d'un agent SNMP, il aurait été découvert lors de la phase de détection, et son interconnexion avec d'autres éléments aurait donc pu être explicitement gérée par l'algorithme.

Description de l'algorithme du Spanning Tree : Le `Spanning Tree Protocol` (STP) est un protocole de gestion de niveau 2, qui fournit des chemins redondants dans un réseau local tout en évitant les boucles de routage. Le protocole STP utilise un algorithme réparti qui sélectionne un pont (commutateur) d'un réseau, dont la connectique physique engendre des liens de secours (soit en paire torsadée, soit en fibre optique), comme la racine d'un arbre associé à la topologie courante. Comme dans les réseaux Ethernet un seul chemin actif peut exister entre deux stations[3], l'installation de liens de secours permet de créer plusieurs chemins actifs entre des stations, ce qui cause inévitablement des boucles dans le réseau.

Lorsque les boucles surviennent, certains commutateurs reconnaissent une même station sur plusieurs ports. Cette situation entraîne des erreurs au niveau de l'algorithme d'expédition et engendre la duplication de trames qui seront

[3]Les opérations du Spanning Tree sont transparentes pour les stations d'extrémités (les feuilles de l'arbre de diffusion, c'est-à-dire PCs, imprimantes, etc..).

expédiées. L'algorithme du Spanning Tree fournit un arbre de circulation de l'information en bloquant, dans son arbre de diffusion, un des chemins de données qui engendre la boucle.

Nous utilisons cet arbre pour la construction de notre topologie.

5.5 Mise en œuvre du service de découverte de la topologie

Le service de la découverte de la topologie est un service qui se décompose en deux agents mobiles, pour obtenir une plus grande efficacité dans la réalisation de l'algorithme. Un premier agent, le `NetNodesFinder` pour la recherche des nœuds ProActive en activité sur le sous-réseau et un second, le `DiscoveryAgent`, agent responsable de la collecte des informations SNMP et de la construction de la topologie du sous-réseau. Ces deux agents mobiles sont instanciés par l'`ItineraryServer` responsable d'un sous-réseau.

5.5.1 Le service de localisation des nœuds ProActive

Ce service est représenté par un agent mobile autonome dans le sens où il possède une activité propre (objet actif selon ProActive). Le `NetNodesFinder` utilise les paramètres qui décrivent le sous-réseau pour itérer sa scrutation de tous les hôtes du sous-réseau qui seraient susceptibles d'accueillir des agents mobiles, c'est-à-dire exécutant un nœud ProActive. Il fournit, dès qu'il en a connaissance, le nom du nouveau nœud au service de mise à disposition. Le `NetNodesFinder` arrête son processus de collecte lorsqu'il a atteint la dernière adresse IP de la plage spécifiée par l'adresse du sous-réseau (`Network Address`) appliquée au masque du sous-réseau (`Network Mask`). Le processus de renouvellement de la liste des nœuds trouvés est automatiquement réalisé toutes les 2 minutes, après la localisation du dernier nœud du sous-réseau. Une optimisation peut être apportée à cet algorithme, en utilisant la technique du protocole `ICMP`, identique à celle de la phase de détection (voir section 5.4.2).

Remarque : La recherche des nœuds ProActive se fonde sur `RMI` pour localiser un `rmiregistry` sur un hôte distant et s'il existe lui demander la liste des nœuds ProActive enregistrés. Si l'hôte distant est protégé par un pare-feu, on subit le `timeout RMI` qui peut être long.

Plus précisément, la recherche des nœuds ProActive se fonde sur `RMI` pour localiser un `rmiregistry` sur un hôte distant et s'il existe lui demander la liste des nœuds ProActive enregistrés. Si l'hôte distant est protégé par un pare-feu, on subit le `timeout RMI` qui peut être long.

5.5.2 Le service de découverte

Le service de découverte est implanté via le `DiscoveryAgent`. Le `DiscoveryAgent` implémente l'algorithme de la construction de la topologie que nous venons de décrire.

Remarque : Suite à la remarque faite précédemment au sujet du `timeout`, on a mis en place deux agents, plutôt qu'un seul : l'un s'occupe uniquement de la localisation, l'autre d'implémenter le service de découverte.

5.5.3 Interaction avec le serveur d'itinéraires

L'`ItineraryServer` est un objet actif mobile qui supervise le fonctionnement des deux objets que nous venons de décrire : le `NetNodesFinder` et le `DiscoveryAgent`. En permanence, l'`ItineraryServer` veille à ce que les services soient les plus du sous-réseau supervisé. Si l'`ItineraryServer` détecte qu'il a la possibilité de se rapprocher du sous-réseau qu'il supervise (grâce à la liste des nœuds ProActive collectée), alors l'`ItineraryServer` utilise la possibilité de migration des trois objets implantant services (lui-même, le `DiscoveryAgent` et le `NetNodesFinder`) afin de les déplacer sur un nœud du sous-réseau.

Par exemple, imaginons les services localisés sur le réseau A (cf. figure 5.17). Le premier cas présente l'exécution des services sur le réseau A, dont toutes les requêtes sont à destination du réseau B. Si l'`ItineraryServer` détecte grâce à la liste des nœuds ProActive qu'il gère, que l'ensemble des services peuvent être déplacés vers le sous-réseau B, alors l'`ItineraryServer` envoie un ordre de migration au `NetNodesFinder` et au `DiscoveryAgent` (2ème cas de la figure).

L'intérêt de cette migration est de permettre plus rapidement l'accomplissement de l'algorithme de construction de la topologie. En effet, se rapprocher du sous-réseau permet d'utiliser la bande passante du réseau local sans avoir à tenir compte de la vitesse du lien entre les réseaux A et B de la figure 5.17.

5.5.4 Topologie de réseaux virtuels

La technologie permet désormais de segmenter les réseaux locaux en une multitudes de réseaux virtuels (*VLAN* pour *Virtual LAN*) qui sont gérés par des commutateurs. On peut définir sur une même classe d'adresse IP, par exemple 192.168.10.0, plusieurs VLANs. Si on se place du côté de la topologie virtuelle, un algorithme effectuant la distinction par VLANs donnera plusieurs topologies sur le même sous-réseau. Nous avons choisi de ne pas faire cette distinction pour obtenir une topologie la plus représentative du sous-réseau, en associant tous les VLANs existant dans la même topologie. Toutefois, nous mettons à disposition les listes des éléments par VLAN dans le sous-réseau. Il est cependant possible d'obtenir, avec notre algorithme de construction de la topologie, une topologie

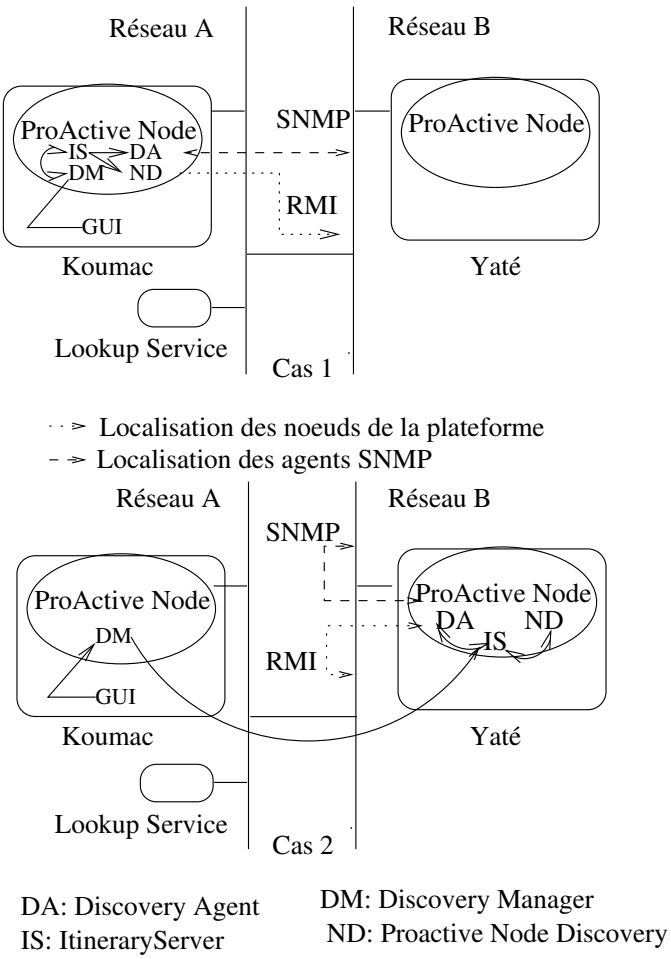

FIG. 5.17 – Schéma d'une migration automatique des services au plus proche du sous-réseau

par VLAN(s) en spécifiant pour les hôtes du réseau qui sont des commutateurs le nom de la communauté SNMP permettant de filter le VLAN désiré.

Par exemple, dans l'exemple présenté par la table 5.1, on utilisera une `SNMP-Destination` avec comme nom de communauté SNMP la valeur :
`public@numero_du_vlan` . On rappelle cependant que le filtre indiqué ici s'applique aux commutateurs de la gamme Cisco. Du matériel d'autres grands constructeurs, comme HP, n'ont pas pu être validés avec notre algorithme.

5.5.5 Evaluation du temps de construction

L'évaluation du temps de construction de la topologie dépend de certains paramètres comme :

– De l'heure de la journée, car dans la journée il y a plus de charge sur le réseau que pendant la nuit

- De la vitesse des interfaces réseau de chacun des éléments sur lesquels on collecte l'information
- De la vitesse de réponse de l'agent SNMP sur lequel on collecte de l'information
- Du nombre de variables SNMP que l'on collectera sur l'agent SNMP
- Du nombre de :
  - Postes de travail connectés, en incluant les serveurs et si quelques uns d'entre eux ont des Agents SNMP
  - Nombre de commutateurs SNMP dans le réseau
  - S'il y a plus d'un routeur

On peut considérer que sur un réseau local, pour lequel un administrateur va devoir intervenir, il va y avoir X éléments connectés. Le temps de détection d'un élément va varier suivant le type d'élément du réseau que l'on va interroger et si ces éléments sont équipés d'un agent SNMP. Plutôt que d'attendre un `TimeOut` de la couche IP, nous vérifions la connectivité de l'élément (`ping`) avant d'essayer d'obtenir plus d'information.

Remarque : Les valeurs qui sont données ici ont été obtenues sur un réseau de production, dont la charge réseau a été augmentée artificiellement, afin d'obtenir des valeurs élevées de collecte des variables SNMP. Ces valeurs ne peuvent pas être appliquées sur d'autres réseaux car elle sont dépendantes de l'architecture même du réseau, néanmoins comme valeurs réalistes.

Considérons donc X le nombre de machines total du réseau. Statistiquement, pour X machines connectées en réseau (cf. table 5.7), nous allons pouvoir dire qu'il y a des routeurs, des commutateurs, des imprimantes en réseau, etc..

Types d'éléments	Nombre	Valeur	Estimation haute du temps de collecte pour un élément de ce type
Postes Clients	X	nombre total	0,5 seconde (Ping)
Routeur	1	au minimum	8 secondes
Commutateurs	5% * X	valeur moyenne	9,5 secondes
Serveurs	6% * X	valeur moyenne	2,5 secondes
Imprimantes en réseau	8% * X	valeur moyenne	1,5 secondes

Tab. 5.7 – Répartition par catégories d'éléments sur un réseau

Notre évaluation du temps de construction de la topologie d'un réseau local peut être estimée à partir de la formule suivante :

Temps de construction = X*0,5s
+ X*5% (nombre moyen de commutateurs SNMP sur ce réseau)*9,5s
+ 1 (routeur)*8s
+ X*8% (nombre imprimantes en réseau)*1,5s
+ X*6% (nombre de serveurs moyens)*2,5s

L'hypothèse est de dire que le trafic moyen ne perturbe pas l'évaluation moyenne du temps de collecte et que ce temps inclut la vitesse de fonctionnement de l'agent SNMP, la vitesse du client, et l'interconnexion des deux éléments dans le même réseau local.

5.5.6 Extension de l'agorithme pour le protocole SNMP V3

Même s'il n'est pas encore totalement déployé dans toutes les infrastructures de réseau, le protocole SNMP V3 est un protocole incontournable car il apporte la sécurité dans les opérations d'administration du réseau. On retrouve dans les architectures de réseau actuelles, de plus en plus d'équipements actifs qui intègrent le protocole SNMP V3, mais aussi une multitude d'équipements actifs fonctionnant encore avec le protocole SNMP V1 (boîtier d'impression, imprimantes en réseau), des éléments qui restent toutefois à administrer avec la première version du protocole SNMP. Cette hétérogénéité nous contraint à proposer une solution de notre algorithme de découverte de la topologie, utilisant les deux variantes de communication du protocole.

Comparativement à la solution exposée dans la section 5.3.2, le seul problème est la méthode de localisation des équipements actifs dans le réseau. En effet, dans le premier algorithme, une méthode dite de broadcast permet de retrouver grâce au protocole SNMP V1 tous les équipements actifs du réseau associé au nom de la communauté SNMP fournie (en général *public* défini dans le descriptif du sous-réseau). Avec cette technique, l'algorithme de construction de la topologie localise de manière aisée tous les équipements actifs. Dans le cas de l'utilisation du protocole SNMP V3, une telle méthode ne peut pas être mise en œuvre. Pour palier au fait que les équipements actifs SNMP V3 ne répondent pas à ces paquets de broadcast, nous proposons de faire ainsi : scrutation des éléments de bordure (par exemple, le routeur par défaut du LAN défini dans le descriptif du sous-réseau), qui permet de lancer itérativement la recherche des autres éléments en activité dans le réseau local.

Successivement et en plusieurs passes, l'ensemble des équipements actifs fonctionnant aussi bien avec le protocole SNMP V1, V2 ou V3 seront ainsi découverts. S'agissant des éléments réseaux conformes au protocole SNMP V3, il est bien évident que le service de découverte possède déjà les données nécessaires qui seront utilisées lorsque l'on voudra converser avec de tels éléments. Par la mise

en œuvre de cette technique, tous les équipements actifs du réseau seront ainsi répertoriés.

Une fois ces éléments découverts, l'algorithme, implémenté par le service de découverte, reste le même, puisque la détermination de la topologie se base sur les mêmes informations que celles obtenues avec la version de protocole non sécurisée. Du coup, les évaluations de performances faites dans la section précédente pour la construction de la topologie du réseau ne seront pas modifiées car dès la phase de démarrage de la construction, les informations listant les équipements actifs et leur protocole de communication auront été obtenus par la modification qui vient d'être présentée.

5.6 Bilan

Dans ce chapitre, nous avons défini les services que nous considérons utiles pour définir une plate-forme pour l'administration système et réseau :

- le service de collecte et d'analyse du réseau qui sert à collecter les données et les services du réseau (nœuds ProActive, agents SNMP, etc..) : ce service est mis en œuvre par le `NetNodesFinder` et le `DiscoveryAgent`
- le service de mise à disposition de toutes les données collectées, organisées selon la catégorie d'administration : une liste pour les agents SNMP et une liste pour les nœuds ProActive : ce service est mis en œuvre par l'`ItineraryServer`
- le service de supervision de la plate-forme qui permet la visualisation de la topologie du réseau, la supervision de tous les services inhérents au fonctionnement de la plate-forme, et pour finir l'aide à la supervision des agents mobiles d'administration (`IC2D` et un outil `ad hoc`) : ce service est mis en œuvre par le `DiscoveryManager`

Pour obtenir la topologie décrivant le réseau à administrer constitué de sous-réseaux, nous avons introduit un objet, de type `SubNetworkDescription`, qui regroupe les informations décrivant un sous-réseau. A partir de ces informations, le service de collecte et d'analyse des données peut construire la topologie d'un sous-réseau. L'algorithme qui nous permet d'obtenir la topologie d'un sous-réseau est donné dans la section 5.4, ainsi que l'interprétation des données collectées pour créer l'arbre n-aire représentant logiquement cette topologie.

Chapitre 6

Programmation d'agents mobiles pour l'administration système et réseau

6.1 Introduction

Dans ce chapitre nous mettons en évidence deux facettes dans la programmation des agents mobiles pour des opérations d'administration système et réseau. La première, basée sur le concept des itinéraires que l'agent mobile doit suivre pour effectuer les tâches programmées. La deuxième basée sur le code de la fonction d'administration à proprement parler pour laquelle nous proposons un cadre générique pour le développement d'agents mobiles d'administration.

Nous commencerons la présentation par l'aspect de la fabrication des itinéraires, représentés par des instances de la classe Java `ItineraryManager` que nous définissons. Cette classe `ItineraryManager` est le point d'articulation entre d'une part la plate-forme sous jacente qui collecte des informations sur le réseau à administrer (cf. chapitre 5) et d'autre part, les agents mobiles qui sont eux en charge des fonctions d'administration sur les éléments du réseau qui leur sont indiqués dans les itinéraires, grâce aux informations collectées.

Pour utiliser nos itinéraires de manière automatique, nous fournissons un modèle de programmation pour un agent (un agent générique), présenté par la classe `Agent`, qu'il suffira de spécialiser pour développer de nouveaux agents mobiles d'administration au sein de notre plate-forme. Toutefois, cette classe `Agent` n'est qu'un modèle que le programmeur n'est pas obligé de suivre pour créer ses propres agents mobiles d'administration. Il s'agit pour nous uniquement de proposer un cadre général de programmation d'agents de notre plate-forme.

6.2 Fabrication et suivi d'itinéraires

6.2.1 Architecture générale

Dans le modèle de migration de ProActive, on trouve juste la définition de ce qu'est un agent mobile et d'une classe permettant de faire suivre un itinéraire à un agent, le `MigrationStrategyManager`. Le `MigrationStrategyManager` est une classe dont une instance permet de gérer de manière automatique les appels de méthodes et les ordres de migration pendant le suivi de l'itinéraire. De plus amples informations seront données sur son fonctionnement dans la section 6.2.3.

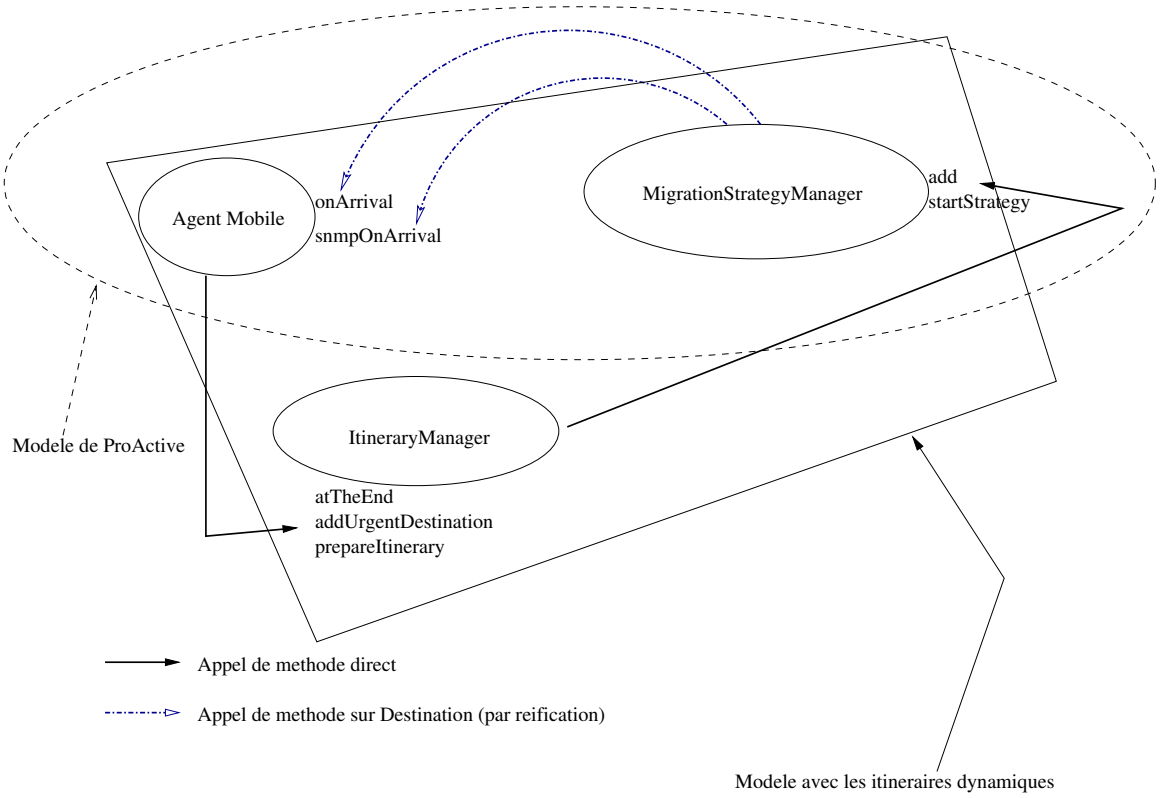

FIG. 6.1 – Interaction entre l'agent mobile, l'ItineraryServer et l'ItineraryManager

Comme on veut faire des itinéraires dynamiques, on a introduit une entité supplémentaire l'`ItineraryManager` qui a pour rôle de récupérer auprès d'un `ItineraryServer` les éléments qui permettront de construire l'itinéraire. La figure 6.1 présente les deux modèles, celui de base de ProActive et les extensions que nous avons apportées en introduisant l'`ItineraryManager`.

L'`ItineraryManager` met à la disposition de l'agent mobile les méthodes publiques suivantes (voir le code figure 6.2) :

– `prepareItinerary`, pour demander la création de l'itinéraire selon le modèle de visite désiré (itinéraire local, distant, etc..)

- `setLastDestination`, pour qu'une méthode particulière soit appelée à la fin de l'itinéraire (par le biais de la `Destination`)
- `startItinerary` pour lancer le suivi de l'itinéraire
- `getCurrentDestination` pour obtenir la `Destination` courante
- `addUrgentDestination`, pour qu'une `Destination` urgente soit prise en compte dans l'itinéraire courant, pour s'occuper de quelque chose de critique, éventuellement dans un autre sous-réseau. Reprise normale de l'itinéraire une fois cette tâche urgente entreprise
- `getOthersItineraryServer` pour obtenir la liste des autres `ItineraryServers` actifs

6.2.2 Fabrication de l'itinéraire

Le fonctionnement général de notre système de fabrication d'itinéraire, implémenté par la classe `ItineraryManager` est décomposé en deux grandes parties. La première partie consiste en la localisation du bon `ItineraryServer`, c'est-à-dire celui qui est responsable de la supervision d'un sous-réseau donné. Pour ce faire, et grâce à la technologie Jini, notre `ItineraryManager` va d'abord récupérer la liste complète de tous les `ItineraryServers` (ces références sont automatiquement obtenues par le Lookup Service). Ainsi, pour obtenir la référence vers l'`ItineraryServer` responsable du bon sous-réseau, il suffit de filtrer la liste résultante et de ne conserver que celui qui est intéressant pour construire la première portion de l'itinéraire (méthode `getLocalItineraryServer`, cf. code figure 6.3). Dès la référence de l'`ItineraryServer` connue, l'`ItineraryManager` collecte la liste des éléments SNMP et la liste des nœuds ProActive détectés par l'algorithme de construction de la topologie pour ce sous-réseau.

La deuxième partie est la partie qui est propre à chaque type de gestionnaire d'itinéraire. Pour chacune des sous-classes de l'`ItineraryManager` la manière dont doit être suivi l'itinéraire est définie. Par exemple, celui-ci peut n'incorporer que des nœuds de la plate-forme, qu'une liste d'agents SNMP à visiter, ou un mélange particulier des deux. Ainsi chaque sous-classe permet de fabriquer un itinéraire en alimentant une liste de `Destinations` représentant les `Destinations` qui doivent être prises en compte lors du suivi de l'itinéraire. Cette liste de `Destinations` est stockée dans le `MigrationStrategyManager` dont la tâche est d'assurer les opérations liées au suivi de l'itinéraire.

6.2.2.1 Un itinéraire avec des éléments de même classe de service

Avec notre modèle et les informations que nous connaissons sur les éléments qui composent le réseau, il est relativement aisé d'implémenter une classe, par exemple `ItineraryManagerForNetworkPrinter` (cf. figure 6.4) qui donnerait un

```
public abstract class ItineraryManager implements java.io.Serializable {
  protected MgtMigrationStrategyManagerImpl mgtMigrationStrategyManager;
  protected MigrationStrategy myItinerary;
  protected ItineraryServerList othersItineraryServer;
  protected Body b; // is the Body of the mobile code
  protected Destination lastDestination;
  private   ArrayList myItineraryServerList=new ArrayList();
  /** Constructor */
  public ItineraryManager() {}
  /** Locate the ItineraryServer and prepare an Itinerary */
  abstract public void prepareItinerary();
  abstract public void prepareItinerary(ItineraryServer itineraryServer) ;
  /** which method to call at the end ?  */
  public void setLastDestination(Destination aDestination) {
    this.lastDestination=aDestination;
  }
  /** start itinerary  */
  public void startItinerary() {
    try {
      mgtMigrationStrategyManager.startStrategy(b);
    } catch (Exception e) {e.printStackTrace();}
  }
  /** Get the current NodeDestination */
  public Destination getCurrentDestination() {
    return mgtMigrationStrategyManager.getCurrentDestination();
  }
  /** Add an Urgent Node to visit */
  public void addUrgentDestination(Destination node) {
    myItinerary.addNext(node);
  }
  /**  return an ItineraryServerList of all ItineraryServers
     found, except current one  */
  public ItineraryServerList getOthersItineraryServer() {
    return othersItineraryServer;
  }
}
```

FIG. 6.2 – Les méthodes publiques de la classe de l'`ItineraryManager`

itinéraire pour n'administrer que des éléments ayant le port `TCP 515`[1] d'ouvert.

Il s'agit uniquement de redéfinir la méthode `prepareItinerary` de la classe `ItineraryManager` pour obtenir l'itinéraire désiré.

6.2.2.2 Un itinéraire couvrant l'ensemble du réseau

Lorsque nous souhaitons bâtir des itinéraires couvrant plusieurs sous-réseaux (cf. figure 6.5), alors la méthode `prepareItinerary` a un fonctionnement particulier. Comme nous voulons des itinéraires les plus à jour possible, la méthode `prepareItinerary` sera appelée plusieurs fois pendant le déroulement du suivi de l'itinéraire. En effet, à chaque fin de sous-itinéraire correspondant à un sous-réseau, la destination `ISDestination` sera insérée pour que l'`ItineraryMana-`

[1]Le numéro de port 515 définissant le service associé à un service d'impression IP (protocole `lpr/lpd` [47]).

```
// create a MigrationStrategyManager instance
private void createMigrationStrategyManager() {
  b = ProActive.getBodyOnThis();
  if (mgtMigrationStrategyManager == null) {
    mgtMigrationStrategyManager = new MgtMigrationStrategyManagerImpl((Migratable) b);
    myItinerary = mgtMigrationStrategyManager.getMigrationStrategy();
  }
}
/** Service lookup Discovery */
protected ArrayList getLookupServices(String services) {
  ServiceLookup sl = new ServiceLookup(services);
  ArrayList listOfService = new ArrayList();
  try {
      sl.waitForLastDiscovery();
      listOfService.addAll(sl.getServices());
  } catch (Exception e) {e.printStackTrace();}
}
/** locate our local ItineraryServer  */
protected ItineraryServer getLocalItineraryServer() {
      IpAddress myAddress = new mgt.ip.InetInfo().getHostIpInfo();
      ItineraryServer localItineraryServer=null;
      for (int i=0;i<myItineraryServerList.size();i++) {
          try {
      ItineraryServerProxy isp =( ItineraryServerProxy) myItineraryServerList.get(i);
      if (!isp.getSubNetworkDescription().isInNetwork(myAddress))
          othersItineraryServer.add(new ISDestination(
              (ItineraryServer)ProActive.lookupActive("ItineraryServer",
                  isp.toString())));
      else
          localItineraryServer = (ItineraryServer)ProActive.lookupActive(
          "mgt.itinerary.ItineraryServer", isp.toString());
      } catch (Exception e) {e.printStackTrace();}
      }
      if (localItineraryServer != null) return localItineraryServer;
  // at this point, no itineraryServer is active for my local network
  // Create one, and wait for data ready
  ArrayList myNetworkList=new ArrayList(getLookupServices("NetworkDescription"));
  SubNetworkDescription subNetDescr=null;
  for (int i=0;i<myNetworkList.size();i++) {
    NetworkDescription NetDescr =( NetworkDescription) myNetworkList.get(i);
    if (NetDescr.getSubNetworkDescription(myAddress) != null)
      SubNetDescr = NetDescr.getSubNetworkDescription(myAddress);
  }
  // try to create an ItineraryServer for local network
  ItineraryServer itineraryServer=null;
  try {
    Node nodeToStart = JiniNodeFactory.getDefaultNode();
    itineraryServer =
      (ItineraryServer) ProActive.newActive("ItineraryServer",null,nodeToStart);
    if (subNetDescr != null)
      itineraryServer.setSubNetworkDescription(subNetDescr);
    else
    System.out.println("No configuration Found. "+
                 Please contact your system administrator !!!");
    return ItineraryServer;
  } catch (Exception e) {e.printStackTrace();}
}
```

FIG. 6.3 – Les méthodes privées ou protégées de la classe de l'`ItineraryManager`

```
Class ItineraryManagerForPrinter extends ItineraryManager{
  public void prepareItinerary (....)

  // Prepare the itinerary
  public void prepareItinerary(ItineraryServer iS) {
      createMigrationStrategyManager()
      iS.getSnmpNodes();
      // éliminer les éléments qui ne conviennent pas
      // c'est-à-dire prendre les éléments qui fournissent le service.
      // Host.getTcpService contient lpr/lpd, port 515

      //parmis ceux qui restent
      SnmpDestination dest = new SnmpDestination(host, community, "printerOnArrival");
      // c'est ici que l'on détermine le nom de la méthode qui sera invoquée :
      // dans ce cas, la méthode appelée "printerOnArrival"
  }
}
```

FIG. 6.4 – Un ItineraryManager pour la collecte de données sur des imprimantes du sous-réseau représenté par un ItineraryServer

ger puisse enrichir l'itinéraire courant. Il y aura un appel de méthode, comme pour une `NodeDestination` qui engendrera l'appel sur la méthode `prepareItinerary` de l'`ItineraryManager`. Ainsi l'`ItineraryManager` enrichira l'itinéraire courant avec les `Destinations` du nouveau sous-réseau à visiter. Et ce, tant qu'il existe des `ItineraryServers` dans la liste obtenue via Jini. Un problème technique est que l'on ne peut pas placer, dès la construction initiale de l'itinéraire, toutes les `ISDestinations` dans l'itinéraire car sinon on ne pourrait plus insérer les éléments correspondant à chaque sous-réseau. On ne place donc à la suite de la liste des éléments du sous-réseau courant que la prochaine destination de type `ISDestination`. Lorsque le `MigrationStrategyManager` déclenche la méthode associée à cette `ISDestination` sur l'agent, cela déclenchera l'exécution de `prepareItinerary` pour le réseau associé à cet `ItineraryServer`. A nouveau, après insertion des éléments de ce réseau (liste récupérée dynamiquement auprès du `ItineraryServer`), on rajoute si besoin la prochaine `ISDestination`, et ainsi de suite. S'il n'y a plus de `ISDestination`, la méthode `prepareItinerary` rajoute, le cas échéant, la destination indiquée par la méthode `setLastDestination`.

Remarque : Toutefois, si l'ItineraryServer n'existe pas pour le réseau cible, son instanciation est initiée afin de pouvoir créer un itinéraire une fois que les informations auront été collectées.

6.2.3 Suivi de l'itinéraire

Le suivi de l'itinéraire est assuré par l'entité `MigrationStrategyManager` (cf. figure 6.6) qui assure donc les opérations de migration de l'agent mobile et les

```
public class ItineraryManagerWholeNetwork
  extends ItineraryManager implements Serializable {
   public ItineraryManagerWholeNetwork() {}
   public void prepareItinerary() {
         ItineraryServer iServer=getLocalItineraryServer();
         prepareItinerary(iServer);
   }
   /** Locate the ItineraryServer and start to migrate */
   public void prepareItinerary(ItineraryServer iServer) {
      proActiveNodes = new ArrayList();
      try {
         ArrayList snmpList = new ArrayList(iServer.getSNMPElements());
         proActiveNodes.addAll(iServer.getProActiveNode());
         createMigrationStrategyManager(); // create a new
         MigrationStrategyManager
         for (int i=0;i<proActiveNodes.size();i++) {
               NodeDestination nd =(NodeDestination)proActiveNodes.get(i)
               myItinerary.add(nd);
               int currentPos =
               snmpList.indexOf(nd.getDestination().getHostName());
               if (currentPos >=0) { // add SNMPDestination for same Host
                  myItinerary.add(snmpList.remove(currentPos));
               }}
         }
         // add others SNMPDestination, host without any Node
         for (int i=0;i<snmpList.size();i++)
           myItinerary.add(snmpList.get(i));
         if (othersItineraryServer.size() >0) {
            // at least another ItineraryServer on whole administrative domain
            // add a ISDestination to call another ItineraryServer
            // the name of called method : prepareItinerary
            myItinerary.add(othersItineraryServer.next());
         }
         else if (lastDestination != null) {
           myItinerary.add(lastDestination);
         }
        } catch (Exception e) {e.printStackTrace();}
   }
}
```

FIG. 6.5 – Itinéraire couvrant tout le réseau : ItineraryManagerWholeNetwork

appels de méthodes associées aux `Destinations`.

Pour chaque `Destination` rencontré pendant le parcours de l'itinéraire, le `MigrationStrategyManager` exécute la fonctionnalité suivante :

- Stockage de la `Destination` courante
- Appel du nom de la méthode contenue dans la `Destination`
- Si `Destination` incluant une migration, alors migration de l'agent mobile

6.3 Code d'administration

6.3.1 Les opérations de base

Nous utilisons pour programmer nos fonctions d'administration de réseau la bibliothèque SNMP fournie par AdventNet [2]. Il est possible d'utiliser l'API

```
public class MgtMigrationStrategyManagerImpl implements
MigrationStrategyManager, MigrationEventListener,
java.io.Serializable  {
  // La destination cournate
  Destination currentDestination = null;
  //Indique que l'objet suit un itinéraire
  private boolean onItinerary;
  // Retourne la destination cournate
  public Destination getCurrentDestination() {
    return currentDestination;
  }
  // méthode pour démarrer le suivi de l'itinéraire
  public void startStrategy( ..) {
    ...
  }
  ...
  // Les appels de méthodes et la migration
  protected void continueStrategy(Body body) throws MigrationException {
    if (! onItinerary) return;
    Destination dest =  migrationStrategy.next();
    currentDestination=dest;
    if (dest == null) { // fin de l'itinéraire
      this.onItinerary = false;
      return;
    }
    methodOnArrival = dest.getMethodName();
    if (dest instanceOf mgt.itinerary.SnmpDestination) {
      executeMethod(body.getReifiedObject(), methodOnArrival);
      continueStrategy(body);
    } else {
      try {
        if ( NodeFactory.isNodeLocal(NodeFactory.getNode(dest.getDestination())) ) {
           executeMethod(body.getReifiedObject(), methodOnArrival);
           continueStrategy(body);
         } else ProActive.migrateTo(body, ie.getDestination(), fifoFirst);
      } catch (Exception e) {e.printStackTrace();}
    }
  }
}
```

FIG. 6.6 – Classe `MgtMigrationStrategyManagerImpl`

fournie par AdventNet afin de collecter des informations sur les agents SNMP. Pour permettre d'écrire des fonctions d'administration plus sophistiquées, nous avons écrit une bibliothèque de plus haut niveau permettant, par exemple, de reproduire une table de routage, une matrice de commutation et d'accéder à ces informations de manière simplifiée.

6.3.1.1 La gestion des traps SNMP

Il est évident que lorsque l'on parle d'un système d'administration système et réseau la plate-forme qui gère le réseau doit être en mesure de prendre en compte les événements qui surviennent sur le réseau. Plus précisément dans notre cas, la récupération des *traps* [88] et le traitement de ces événements. La gestion et la compréhension de ces traps SNMP imposent un long travail afin de mettre à

disposition dans la plate-forme d'administration système et réseau toutes les *OID SNMP* correspondant chacun à une trap d'un équipement.

```
public boolean callback(SnmpSession session,SnmpPDU pdu, int requestID){
  // check trap version
  if(pdu == null) return false;
  if (pdu.getCommand() == api.TRP_REQ_MSG) {
    System.out.println("Trap received from: "
      +pdu.getAddress() +", community: " + pdu.getCommunity());
    System.out.println("Enterprise: " + pdu.getEnterprise());
    System.out.println("Agent: " + pdu.getAgentAddress());
    System.out.println("TRAP_TYPE: " + pdu.getTrapType());
    System.out.println("SPECIFIC NUMBER: " + pdu.getSpecificType());
    System.out.println("Time: " + pdu.getUpTime()+"\nVARBINDS:");
    // print varbinds
    for (Enumeration e = pdu.getVariableBindings().elements();e.hasMoreElements();)
      System.out.println(((SnmpVarBind) e.nextElement()).toTagString());
  }
  else
    System.err.println("Non trap PDU received.");

  System.out.println(""); // a blank line between traps
    try {
      // create a mobile agent and sent it to see what's happen
      AgentTrap agtrap = (AgentTrap) ProActive.newActive("AgentTrap",null);
      agtrap.setTrapFrom(pdu.getAddress().toString(),
         pdu.getCommunity().toString(), pdu.getEnterprise().toString());
      agtrap.check();
     } catch (Exception e) {e.printStackTrace();}
       return true; // API CallBack of AdventNet
  }
```

FIG. 6.7 – Exemple de code inspiré par la gestion des traps de AdventNet pour instancier un agent mobile de gestion de traps

```
public class AgentTrap extends Agent implements  java.io.Serializable {
  ArrayList nodeRessource = new ArrayList();
  ArrayList arpTable = new ArrayList();
  String ipAddress, community, trapOID;

  public AgentTrap() {
      super();
  }
  // method call by a trap listener
  public void setTrapFrom(String ipAddress, String community, String trapOID) {
      this.ipAddress=ipAddress;
      this.community=community;
      this.trapOID=trapOID;
  }
  // go and check what's happening
  public void check() {
      System.out.println("A trap is received from "+ipAddress);
  }
}
```

FIG. 6.8 – Un agent générique permettant de gérer une trap SNMP

Toutefois, et à titre d'exemple, le code (cf. figure 6.7) basé sur la gestion des traps SNMP de AdventNet, permet simplement de capturer la *trap* (fonction `callback` et d'instancier ensuite un agent mobile (cf. figure 6.8) qui aurait pour travail sa gestion. Dans notre exemple, l'agent mobile affiche la provenance de la trap SNMP, mais pourrait tout aussi bien déclencher un ou plusieurs autres agents mobiles pour effectuer une tâche d'administration liée à cet événement. Pendant la phase de traitement de cette *trap* par l'agent mobile, si un autre événement survient, alors les données relatives à cette *trap* peuvent être transmises à l'agent mobile. En effet, le système de communication de ProActive permet d'échanger des messages avec un agent mobile pendant son processus de migration, quelque soit l'emplacement de celui-ci. Par un tel système, l'agent mobile responsable de la gestion des *traps* pourra corréler les différents événements si nécessaire.

6.3.1.2 Appels systèmes directs

Pour programmer des fonctions d'administration de systèmes, nous avons selon les cas utilisé les appels systèmes directs, via l'interface native de Java (Java Native Interface, JNI), utilisé l'API des entrées/sorties de Java, ou utilisé l'API *System* de Java.

Nous avons essayé de masquer, par une bibliothèque de fonctions de plus haut niveau, la complexité et l'hétérogénéité des systèmes (par exemple pour obtenir les ressources d'un système). Sachant que, même les opérations d'administration système (de consultation, surveillance) sont de plus en plus réalisables via uniquement l'interrogation d'agents SNMP, il sera dès lors possible de consulter la MIB des agents SNMP pour obtenir le même type d'information. Par exemple, les ressources propres d'un système peuvent être accessibles par la *Host* MIB [36].

6.3.2 Utilisation des opérations de base dans le code de l'agent

Il suffit d'hériter du squelette de la classe `Agent`, en se focalisant sur les méthodes qui sont déclenchées automatiquement et qui implantent le code fonctionnel : tout particulièrement `onArrival`, `snmpOnArrival`, c'est-à-dire les méthodes qui sont déclenchées par le `MigrationStrategyManager`.

6.3.2.1 Squelette de l'agent

Le squelette de l'`Agent` (figure 6.9) définit les méthodes qui sont nécessaires pour utiliser cette définition d'un agent mobile avec des itinéraires dynamiques. Les méthodes `onArrival` et `snmpOnArrival` définies permettent seulement de capturer la destination courante dans une variable afin d'affranchir le programmeur de cette étape et elles devront bien sûr être personnalisées dans les classes filles. Quant aux méthodes liées à la préparation et au démarrage de l'itinéraire,

```
public abstract class Agent java.io.Serializable {
  // the Destinations
  private ItineraryManager itiManager; // my ItineraryManager
  private ArrayList visitedNodes; // collected visited nodes
  // constructor
  public Agent() {}
  public ItineraryManager getItineraryManager() {
    // return the ItineraryManager
    return itiManager;
  }
  public ArrayList getVisitedNodes() {
    // return an ArrayList of all Visited Nodes
    return visitedNodes;
  }
  // what to do for a NodeDestination
  public void onArrival() {
    NodeDestination nDest = (NodeDestination)itiManager.getcurrentDestination();
    visitedNodes.add(dest.toString());
  }
  // what to do for a SnmpDestination
  public void snmpOnArrival() {
    SnmpDestination snmpDest = (SnmpDestination)itiManager.getcurrentDestination();
    visitedNodes.add(dest.toString());
  }
  // Prepare the itinerary
  public void prepareItinerary(ItineraryManager itiManager) {
    // Prepare an ItineraryManager
    this.itiManager=itiManager;
    itiManager.prepareItinerary();
  }
  // Prepare the itinerary
  public void prepareItinerary(ItineraryManager itiManager, ItineraryServer iServer) {
    // Prepare an ItineraryManager using ItineraryServer iServer
    this.itiManager=itiManager;
    itiManager.prepareItinerary(iServer);
  }

  // start to follow the itinerary
  public void startItinerary() {
    itiManager.startItinerary();
  }
  // add an urgent Destination to visit
  public void addUrgentDestination(Destination destination) {
    // the next Destination will be the new one
    itiManager.addUrgentDestination(destination);
  }
  // what to do at the end of itinerary, if necessary
  public void setLastDestination(Destination aDestination) {
    // add a Destination to visit at the end of the current itinerary
    itiManager.setLastDestination(aDestination);
  }
  // what to do at the end of itinerary, if necessary
  public void setLastDestination(String methodName, Node homeNode) {
    itiManager.setLastDestination(
    new NodeDestination(homeNode.getNodeInformation().getName(),methodName));
  }


}// end of class Agent
```

FIG. 6.9 – Classe abstraite de l'Agent

elles font appel aux méthodes d'une instance de la classe `ItineraryManager` associées à l'agent : `prepareItinerary(...)` pour préparer l'itinéraire selon le modèle
d'`ItineraryManager` choisi, `startItinerary()` pour démarrer le suivi de l'itinéraire, `addUrgentDestination(..)` pour prendre en compte une destination à traiter en urgence quitte à dérouter l'agent de son réseau et `setLastDestination(...)` pour faire exécuter une tâche en fin de parcours de l'itinéraire. Notons que c'est l'instanciation d'un type d'`ItineraryManager` voulu qui permet à l'agent mobile de savoir quel type d'itinéraire il devra suivre mais que l'itinéraire à proprement parler est créé dynamiquement par l'agent mobile lui-même (par l'appel à `prepareItinerary`).

6.3.2.2 Exemple d'utilisation

```
public class AgentMix extends Agent java.io.Serializable {
  ArrayList AllMemInfo  = new ArrayList();
  ArrayList AllArplTable = new ArrayList();
  public AgentMix() {}
  public void onArrival() {  // override Agent.onArrival
      super.onArrival();
      SystemResources sr = new SystemResources();
      AllMemInfo.add(nodeDest.getDestination());
      // Gets TOTAL MEM, USED MEM, FREE MEM into a String
      AllMemInfo.add(sr.toString());
  }
  public void snmpOnArrival() {  // overrides Agent.snmpOnArrival
      super.snmpOnArrival();
      // gets the SNMP MIB Table ip.ipNetToMediaTable
      AllArplTable.addAll(new IpMacTable(snmpDest).getArpTable());
  }
  // Display the collected Data
  public void displayData() {
      // displays the collected data
      System.out.println("Data collected : "
          +AllMemInfo+"\n"+AllArplTable);
  }
  // return partial results
  public ArrayList getPartialResult() {
      return AllMemInfo;
  }
}
```

FIG. 6.10 – Un agent mixte, récupérant les ressources du système sur chaque nœud et la table ARP de chaque Agent SNMP

L'`AgentMix` (figure 6.10) est un exemple d'implémentation d'un agent mobile effectuant une tâche d'administration pour les nœuds et une collecte de données pour les agents SNMP. Pour l'écrire, nous avons étendu la classe `Agent` et décrit les méthodes qui nous intéressaient : `onArrival` pour la collecte de la charge des ressources du système et `snmpOnArrival` pour la collecte de la table ARP de l'agent SNMP spécifié dans l'itinéraire. Nous utilisons pour instancier l'`AgentMix`

une instance de la classe `Launch` (cf. figure 6.11). Cette classe fixe les modalités du fonctionnement de l'`AgentMix`, en fournissant à l'agent mobile l'itinéraire `ItineraryManagerLocalNetwork` et en demandant à l'agent mobile d'exécuter à la fin de son itinéraire, la méthode `displayData()` qui sera spécifiée lors de la création de l'agent mobile par l'appel de la méthode `setLastDestination(..)`. Puis cette classe permet de faire autre chose et de temps en temps si nécessaire, d'aller demander à l'agent mobile ses résultats intermédiaires.

```
import org.objectweb.proactive.ProActive;
import org.objectweb.proactive.core.node.NodeFactory;
import AgentMix;
public class Launch {
  public static void main(String args[]) {
    try {
      AgentMix mobileAgent =
        (AgentMix)ProActive.newActive("AgentMix", null);
      mobileAgent.prepareItinerary(new ItineraryManagerLocalNetwork());
      // NodeFactory.getDefaultNode is the ProActive Node on current JVM
      mobileAgent.setLastDestination("displayData", NodeFactory.getDefaultNode());
      mobileAgent.startItinerary();
      // do a job
      ArrayList partialResult=mobileAgent.getPartialResult();
    } catch (Exception e) { e.printStackTrace(); }
  } // main
}
```

FIG. 6.11 – Code du lanceur de l'AgentMix

6.4 Extensibilité du modèle de développement

6.4.1 Modalités de l'extension

Le cadre de développement des agents mobiles que nous venons d'exposer permet d'écrire facilement de nouveaux agents mobiles qui utiliseraient notre bibliothèque d'itinéraires dynamiques. C'est ce que nous avons expliqué dans la section précédente. Ce qu'il serait souhaitable de pouvoir faire, c'est d'étendre le cadre de développement lui-même. C'est la possibilité de pouvoir définir de nouveaux types de `Destinations`, de nouveaux types d'itinéraires prenant en compte ces nouvelles `Destinations`, et des agents mobiles dont les méthodes à déclencher pour les destinations d'un itinéraire puissent avoir des noms quelconques. Pour cela, nous présentons dans la section suivante comment il est possible d'étendre notre modèle afin que le nom de méthode d'administration à faire exécuter par un agent mobile est paramétrable, et comment prendre en compte un nouveau type de `Destination`.

Puis, pour étendre notre modèle a d'autres types de destinations, il faut bien sur définir ce nouveau type puis, il suffit d'étendre l'`ItineraryManager` en faisant une classe d'itinéraire ad hoc prenant en charge l'organisation de ces destinations

pour atteindre l'objectif fixé. En terme de gestion d'itinéraire, il sera nécessaire que ce nouvel `ItineraryManager` sache préparer et démarrer le suivi de l'itinéraire.

Pour associer un agent mobile d'administration avec le système des itinéraires que nous avons développé (`ItineraryManager` et les classes dérivées), il suffit juste que le code de l'agent mobile définisse les méthodes qui seront appelées pendant le suivi de l'itinéraire.

Il n'est pas obligatoire de réutiliser les méthodes que l'on a défini dans la classe `Agent` (cf. figure 6.9) (méthodes `onArrival`, `snmpOnArrival`), puisque le nom de ces méthodes, bien que fixe à une valeur initiale par défaut dans toute classe Destination, peut être redéfinie si besoin est, lors de la preparation de l'itinéraire.

6.4.2 Exemple : prise en compte d'un nouveau type de destination

Pour illustrer l'extensibilité de notre modèle d'itinéraire et de programmation des agents mobiles, nous présentons un agent mobile qui effectuerait une tâche d'administration sur des équipements actifs supportant le protocole SNMP V3. La figure 6.12 présente le code de l'agent mobile avec l'implémentation de la méthode qui sera invoquée (que l'on a appelée `secureV3OnArrival` à la place du nom de la méthode définie par défaut qui est `snmpV3OnArrival`). La classe dérivée de l'`ItineraryManager`, l'`ItineraryManagerSNMPV3` permet de construire un itinéraire supportant le nouveau type de destination, la `SNMPV3Destination` (figure 6.13).

On a cependant une restriction qui est que seuls les éléments définis explicitement (via une GUI ad hoc alimentant les données du service de description d'un sous-réseau, cf. section 5.3.2.1) comme étant de type `SNMPV3Destination` pourront être considérés de ce type. En effet, une détection des équipements actifs supportant le protocole SNMP V3 n'est pas aisée, car les informations concernant la sécurité (par exemple *username*, *password*, etc...) ne peuvent pas être découvertes automatiquement. C'est pour cette raison que le service de découverte d'un sous-réseau est conçu pour découvrir des éléments SNMP V1, des nœuds ProActive, et qu'une extension de notre algorithme de découverte permet aisément de prendre en compte les autres éléments du réseau utilisant les améliorations du protocole SNMP (SNMP V2c et SNMP V3).

6.4.2.1 Sécurisation des échanges

Dans le modèle d'exécution des tâches écrites en Java ProActive, nous ne faisons pas intervenir un modèle de sécurité parce que nous avons considéré que les réseaux de l'entreprise étaient directement inter-connectés, sans passer par un opérateur. Toutefois, avec la dispersion de plus en plus grande des réseaux au sein du réseau de l'entreprise, il peut être préférable d'utiliser la version sécurisée de la

```
Class AgentSnmpV3 extends Agent implements java.io.Serializable {
 private ItineraryManager itiManager;
  public AgentSnmpV3() {}
  // what to do for a NodeDestination
  public void onArrival() { }
  // what to do for a SnmpDestination
  public void secureV3OnArrival() {
    // on récupère la destination SnmpV3 dans la variable snmpDest
    super.snmpOnArrival();
    mgt.snmp.SystemDescr sysDescrV3= new mgt.snmp.SystemDescr();
    sysDescrV3.setSystemDescr("v3",snmpDest.getUsername(), snmpDest.getAuthPassword(),
       snmpDest.getAuthProtocol(),snmpDest.getPrivPassword());
  }
  // Prepare the itinerary
  public void prepareItinerary(ItineraryManager itiManager) {
    // Prepare an ItineraryManager
    this.itiManager=itiManager;
    itiManager.prepareItinerary(itineraryServerHome);
  }
}// end of class AgentSnmpV3
```

FIG. 6.12 – L'`AgentSnmpV3` qui dialogue en SNMP V3 avec des équipements actifs

```
import mgt.itinerary.SnmpDestination;
Class SnmpV3Destination extends SnmpDestination {
     String username; // principal username
     String authPassword; // authentication password
     String authProtocol; // authentication protocol : MD5/SHA
     String privPassword; // privacy control password
     String contextName; // context use for snmp v3 pdu
     int portNumber=161;
     String methodOnArrival="snmpV3OnArrival"; // default method name
     public SnmpV3Destination(String host, String username, String authPassword,
          String authProtocol, String privPassword) {
      ....
     }
     ...
     // gets values for Snmp V3
     public String getUserName() ..
     public String getAuthPassword()..
     public String getAuthProtocol()..
     public String getPrivPassword()..
     public String getContextName()..
     public int getPortNumber()..
}
```

FIG. 6.13 – La définition d'une destination pour le protocole SNMP V3

plate-forme ProActive [6] et de ne permettre l'accès aux données sécurisées qu'aux agents mobiles disposant des droits nécessaires. Regardons à présent comment exécuter une fonction d'administration pour un élément SNMP V3 qui requiert que l'agent dispose des informations sur le nom d'utilisateur et le mot de passe pour le compte duquel il s'exécute. Nous supposons ainsi que ls agents mobiles se déplacent sur des nœuds sécurisés ; leur migration et leur communication peuvent elle être cryptées. Nous avons imaginé un processus d'échange de clef

```
Class ItineraryManagerSNMPV3 implements ItineraryManager{
  // Prepare the itinerary
  public void prepareItinerary(ItineraryServer iServer) {
      ArrayList snmpV3List = iServer.getSnmpElements();
      createMigrationStrategyManager();
      for (int i=0;i<snmpV3List.size();i++) {
           // prendre les éléments SNMP V3
           if (snmp3List.get(i) instanceof SNMPV3Destination) {
           // ceux qui ont été définis explicitement étant du SNMP V3
           // via la GUI de saisie
             SNMPV3Destination destV3=SNMPV3Destination(snmpV3List.get(i));
             destV3.setMethodName("secureV3OnArrival");
           // on change le nom de la méthode de la destination V3, qui est
           // par defaut : snmpV3OnArrival
           // c'est ici que l'on détermine le nom de la méthode qui
           // devra être appelée à "l'arrivée" sur chaque élément
             myItinerary.add(destV3);
          }
      }
  }
}
```

FIG. 6.14 – Un exemple d'un ItineraryManager pour des destinations du type SNMP V3

entre les services principaux de notre plate-forme pour que nos agents mobiles puissent s'authentifier et récupérer des informations confidentielles. Pour ce faire, on introduit un service supplémentaire, le service de sécurité, que l'on suppose être infaillible.

Le processus de mise à disposition des clefs privées est le suivant : le service de sécurité est lancé avec la connaissance de la clef privée, afin de pouvoir authentifier les agents mobiles qui souhaitent avoir les informations confidentielles sur les accès sécurisés en SNMP V3. Chaque agent mobile qui serait créé depuis la station d'administration et qui doit assurer des communications sécurisées en SNMP V3, se voit associer à sa création la clef privée qui lui est fourni par la station d'administration (elle même supposée totalement sécurisée). A chacun de ses déplacements, c'est-à-dire pendant le suivi d'un itinéraire, si un agent mobile a besoin de converser en utilisant le protocole SNMP V3, alors il utilisera sa clef privée pour contacter le service de sécurité. L'agent mobile obtiendra alors les informations confidentielles qui permettront la connexion avec l'agent SNMP V3 de l'équipement actif. Ainsi, à chaque accès sur un équipement actif compatible avec le protocole SNMP V3, la séquence suivante se déroule :

- l'agent mobile muni de sa clef privée contacte le service de sécurité et s'authentifie,
- le service de sécurité retourne à l'agent mobile authentifié, les informations SNMP V3 nécessaires à l'établissement de la session sécurisée vers l'agent SNMP.

Par cette méthode, si un agent mobile est capturé pendant le suivi de son itinéraire (et si bien sûr l'agent est crypté à chaque migration afin que l'on ne puisse pas

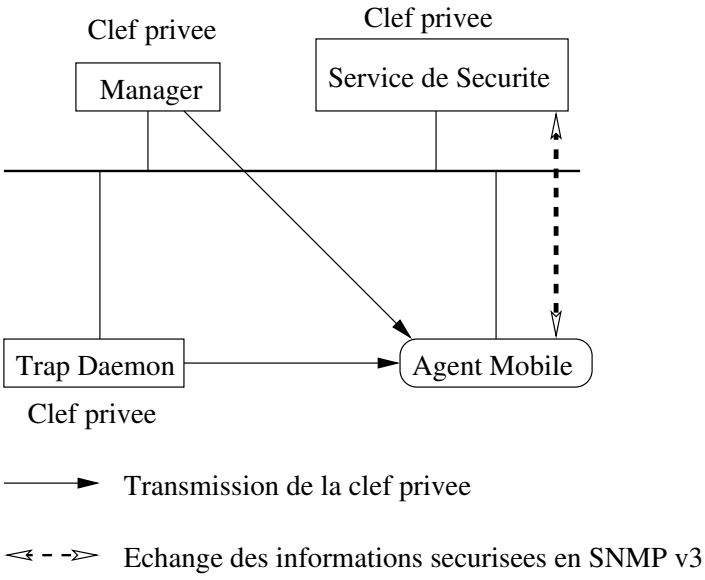

FIG. 6.15 – Principe de sécurisation

découvrir sa clef privée), il devient impossible d'accéder aux agents SNMP V3. Ainsi la sécurité du réseau, et principalement sa configuration, est garantie.

Dans le cas où une alerte est capturée par le service de gestion des *traps* SNMP, et que la réaction consiste à déclencher un agent mobile, il faut que l'agent mobile puisse obtenir les informations concernant les équipements actifs dont les accès sont sécurisés. Pour cela, le service de gestion des *traps* SNMP est lancé en portant à sa connaissance la clef privée permettant l'authentification des agents mobiles auprès du service de sécurité. Lorsque qu'une alerte est capturée, l'agent mobile sera lancé avec cette clef privée afin de pouvoir s'authentifier, comme dans le cas où il est lancé à partir de la station d'administration.

En résumé, puisque dans un itinéraire une `Destination` SNMP V3 peut être incorporée, il s'agit dès lors pour l'agent mobile d'établir une connexion sécurisée vers l'équipement actif. Pour réaliser cela, l'agent mobile contactera le *service de sécurité* pour obtenir les informations nécessaires et établira ensuite la connexion avec l'équipement actif.

6.4.3 Exemple : fabrication d'un nouveau type d'itinéraire parallèle

On va illustrer ici le fait qu'on est capable d'initier la création d'agents mobiles ayant leur propre itinéraire lors de l'exécution de méthode associée à une `Destination` indiquée dans l'itinéraire d'un autre agent mobile. Ceci servira donc d'exemple à l'utilisation de deux types possibles de `Destination` que l'on peut décider de rajouter par extension : `CloneDestination` et `RendezVousDestination`.

L'exemple consistera à administrer tous les systèmes et les éléments du réseau,

de tous les sous-réseaux, en parallèle. L'agent principal aura aussi en charge d'administrer les éléments de son propre sous-réseau.

L'AgentClone (AC, cf. code 6.20) qui joue le rôle de l'agent principal démarre avec un `ItineraryManagerParalleleWholeNetwork` (IMPWN, cf. code 6.16). La préparation de l'itinéraire consiste à :

```
public class ItineraryManagerParallelWholeNetwork extends ItineraryManager
implements java.io.Serializable{

  /** Constructor */
  public ItineraryManagerParallelWholeNetwork() {   super();  }
  public void prepareItinerary() {
     prepareItinerary((ItineraryServer)null);
  }
  public void prepareItinerary(ItineraryServer iServer) {
    RendezVousDestination rdvDestination;
    try {
      if (iServer == null) {
        iServer=getLocalItineraryServer();
      }
      // obj contains a reference onto the local Itinerary Server
      ArrayList proActiveNodes = iServer.getProActiveNode();
      ArrayList snmpList = new ArrayList(iServer.getSNMPElements());
      if (proActiveNodes.isEmpty()) return; // nothing to do, may be an error
      createMigrationStrategyManager();
      // prepare rendez-vous
      int lastIndex =proActiveNodes.size() -1;
      rdvDestination = new RendezVousDestination((Destination)proActiveNodes.get(lastIndex));
      rdvDestination.setMasterAgent((Agent)ProActive.getStubOnThis());
      // to synchronize secondary agents
      NodeDestination nd =
        new NodeDestination(NodeFactory.getDefaultNode().getNodeInformation().getURL(),"");
      // A clone destination on current Node, and give it
      // the list of others ItineraryServers
      CloneDestination cDestination =
       new CloneDestination(nd,othersItineraryServer,rdvDestination);
      myItinerary.add(cDestination);
      for (int i=0;i<proActiveNodes.size() ;i++) {
        NodeDestination nd =(NodeDestination)proActiveNodes.get(i)
        myItinerary.add(nd);
        int currentPos = snmpList.indexOf(nd.getDestination().getHostName());
        if (currentPos >=0)  // add SNMPDestination for same Host
           myItinerary.add(snmpList.remove(currentPos));
}
      // add others SNMPDestination, host without a Node
         for (int i=0;i<snmpList.size();i++)
           myItinerary.add(snmpList.get(i));
      // add RendezVous at end of current Itinerary byt modified method named
      rdvDestination.setMethodName("rendezVousMaster"); // for master agent only
      myItinerary.add(rdvDestination);
    } catch (Exception e) {e.printStackTrace();
    }
  }
}
```

FIG. 6.16 – Classe de l'`ItineraryManagerParalleleWholeNetwork`

– contacter l'`ItineraryServer` (IS) local afin de récupérer la liste des nœuds ProActive.

– créer une `CloneDestination` (CD, cf. code 6.17) pour que l'AC, à son arrivée sur une telle destination, se mette à préparer le nombre d'agents mobiles secondaires nécessaire.
– faire en sorte que le premier élément de l'itinéraire de l'AC soit une `CloneDestination` et les suivants les `NodeDestinations` obtenus via l'ItineraryServer local.
– fixer la dernière destination (`RendezVousDestination`, cf. code figure 6.18) comme celle qui permettra de se synchroniser avec tous les agents secondaires. La méthode appelée pour l'AC sera `rendezVousMaster` qui permet de suspendre le suivi de l'itinéraire en vue de se synchroniser avec les agents secondaires.

```
public class CloneDestination implements java.io.Serializable,
   org.objectweb.proactive.ext.migration.Destination{
private String host, methodName = "cloneOnArrival";
private Destination dest=null;
private int numberOfAgents=0; // Number of agents to duplicate
private DestinationList lastItinerary;
RendezVousDestination rdvDestination;

  public CloneDestination(Destination dest,  DestinationList lastItinerary) {
this.dest=dest;
this.numberOfAgents=lastItinerary.size();
this.lastItinerary=lastItinerary;
  }
  public CloneDestination(Destination dest,  DestinationList lastItinerary,
    RendezVousDestination rdvDestination) {
this.dest=dest;
if (lastItinerary!= null) this.numberOfAgents=lastItinerary.size();
else  this.numberOfAgents=-1;;
this.lastItinerary=lastItinerary;
this.rdvDestination = rdvDestination;
  }
  // return the next part of current itinerary, to be use by clone agents
  public DestinationList getLastItinerary() {
return lastItinerary;
  }
  // return RendezVous Destination
  public RendezVousDestination getRendezVousDestination() {
return rdvDestination;
  }
  // number of agents to wait if necessary
  public int getNumberOfAgent() { return numberOfAgents;
  }
  /** Return method name to execute  */
  public String getMethodName() {
return methodName;
  }
  public Destination getCloneDestination() {
return dest;
  }
}
```

FIG. 6.17 – Classe de la `CloneDestination`

Sur appel de la méthode `cloneOnArrival` liée à la destination de type `Clone-`

`Destination` les actions suivantes sont exécutées :

- On récupère la `RendezVousDestination` (qui est en fait aussi stockée dans la `CloneDestination`)
- On récupère le nombre d'`ItineraryServers`, ce qui donne le nombre d'instances d'`AgentGeneric` (AG, cf.code 6.19) qui doivent être créés.
- Pour chaque élément de la liste (liste des `ItineraryServers`)
  - le master agent (AC) créé un AG avec un `ItineraryManagerLocalNetwork` (pour administrer tous ses éléments, nœuds ProActive et agents SNMP).
  - On fixe la dernière `Destination` comme étant de type `RendezVousDestination` et on demande la préparation de l'itinéraire via l'`ItineraryServer` correspondant. On a fait en sorte que dans cette `RendezVousDestination` on ait une référence vers l'agent principal (ici l'`AC`) afin que les agents secondaires puissent le contacter. Cette `RendezVousDestination` permettra l'appel de la méthode `rendezVousPoint` pour les agents secondaires.
  - L'AG démarre son itinéraire. Sa dernière Destination sera donc la `RendezVousDestination`.
  - Fin de Pour....

Dès que les AGs arrivent sur leur `RendezVousDestination`, ils se synchronisent avec l'AC par l'appel de la méthode `rendezVousPoint()` associée à cette destination, méthode qui est définie dans le corps de l'AC. Cette méthode décompte le nombre d'AG fils qui l'ont appelée au moment de ce rendez-vous. Chaque `AgentGeneric` appelle la méthode `collectData` de l'AC pour transmettre les données collectées.

6.5 Bilan

Dans ce chapitre nous avons présenté la mise en œuvre de notre système de gestion d'itinéraire, basé sur la classe `ItineraryManager`. Ce gestionnaire d'itinéraire se charge de récupérer les informations mises à disposition par un ou plusieurs `ItineraryServers`, qui sont quant à eux localisés sur différents sous-réseaux du réseau à administrer. Avec ces informations, nous avons pu construire différents types d'itinéraires adaptés à des opérations d'administration :

- pour le sous-réseau, comprenant les équipements actifs et les nœuds ProActive
- pour l'ensemble du réseau, c'est-à-dire l'union des itinéraires possibles par sous-réseau
- par type de service fourni par les éléments du réseau, comme par exemple un itinéraire pour toutes les imprimantes

Nous avons aussi défini le cadre de programmation des agents mobiles dans le contexte de l'administration système et réseau en définissant une classe générique

```
public class RendezVousDestination implements java.io.Serializable,
     org.objectweb.proactive.ext.migration.Destination {
private String host;
private String methodName = "rendezVousPoint";
private Destination lastDestination;
private Agent masterAgent;

public RendezVousDestination(Destination lastDestination) {
this.lastDestination=lastDestination;
methodName=lastDestination.getMethodName();
}
public String toString() {
return " rendezvous point "+lastDestination.getDestination());
}

public void setMasterAgent(Agent masterAgent) {
this.masterAgent=masterAgent;
}

public Agent getMasterAgent() {
return masterAgent;
}

/**Return the snmp method to execute */
public String getMethodName() {
return lastDestination.getMethodName();
}

/**  Return current destination as a string */

public String getDestination() {
return lastDestination.getDestination();
}
// get rendezvous Destination
public Destination getRendezVous() { return lastDestination;}
// set a rendezvous
public void setRendezVous(Destination lastDestination) {
this.lastDestination=lastDestination;
}
}
```

FIG. 6.18 – Classe de la `RendezVousDestination`

`Agent` comprenant toutes les fonctionnalités liées à la construction de l'itinéraire, à son suivi ainsi que les appels de méthodes permettant d'effectuer une opération d'administration selon le type d'élément (basé sur la hiérarchie des `Destinations` que nous avons construite).

En respectant la méthodologie prônée par notre cadre de développement, nous avons mis en évidence la simplicité de son extension, qui passe par la création de nouveaux types de destinations, de nouveaux types d'itinéraires.

Le chapitre suivant présente les évaluations que nous avons réalisées sur notre plate-forme en utilisant notre modèle de programmation.

```
public class AgentGeneric extends Agent implements  java.io.Serializable {
  ArrayList nodeRessource = new ArrayList();
  ArrayList AllArplTable = new ArrayList();

  public AgentGeneric() {
    super();
  }
  public void setItineraryManager(ItineraryManager itineraryManager) {
    this.itineraryManager=itineraryManager;
  }
  public void onArrival() {
    super.onArrival();
    SystemResources sr = new SystemResources();
    nodeRessource.add(nodeDest);
    nodeRessource.add(sr.getSystemOS());
    // Gets TOTAL MEM, USED MEM, FREE MEM into a String
    nodeRessource.add(sr.toString());
  }
  public void snmpOnArrival() {  // overrides Agent.snmpOnArrival
      super.snmpOnArrival();
      // gets the SNMP MIB Table ip.ipNetToMediaTable
      AllArplTable.addAll(new IpMacTable(snmpDest).getArpTable());
  }
  public void rendezVousPoint() {
    RendezVousDestination rvDestination =
    (RendezVousDestination)itiManager.getCurrentDestination();
    ((AgentClone)rvDestination.getMasterAgent()).rendezVousPoint();
    ((AgentClone)rvDestination.getMasterAgent()).collectData(nodeRessource);
    ((AgentClone)rvDestination.getMasterAgent()).collectData(AllArplTable);
  }

  public void displayData() {
    System.out.println("Data collected : "+nodeRessource);
  }
}
```

FIG. 6.19 – Classe abstraite de l'AgentGeneric - agent secondaire

```
public class AgentClone extends Agent implements java.io.Serializable {
  ArrayList nodeRessource = new ArrayList();
  ArrayList AllArplTable = new ArrayList();
  ArrayList listAgentsData = new ArrayList();
  private int nbAgentToWait=0;
  public AgentClone() {
    super();
  }
  // master agent suspend it's own itinerary
  public void rendezVousMaster() {
    try { getItineraryManager().suspendItinerary();
    } catch (Exception e) {}
  }
  // method for RendezVousDestination
  public void rendezVousPoint() {
    System.out.println("MASTER AGENT: A child agent is calling me");
    if (nbAgentToWait >0) {
      nbAgentToWait--;
    }
    try {
      if (nbAgentToWait == 0) {
        System.out.println("Master agent can go to the next Destination");
        getItineraryManager().resumeItinerary();
      }
    } catch (Exception e) {e.printStackTrace();}
  }
  // method called by sub Agents
  public void collectData(ArrayList listOfData) {
    listAgentsData.addAll(listOfData);
  }
  public void cloneOnArrival() {
    CloneDestination cloneDest =
       ((CloneDestination)itiManager.getCurrentDestination());
    RendezVousDestination rdvDestination=cloneDest.getRendezVousDestination();
    DestinationList allItineraryServer = cloneDest.getRestOfTheItinerary();
    int nbItineraryServer = allItineraryServer.size();
    nbAgentToWait=nbItineraryServer;
    try {
      if (nbItineraryServer > 0) {
        // create one agent per nodes
        for (int i=0;i<nbItineraryServer;i++) {
          AgentGeneric mobileAgent =
             (AgentGeneric)ProActive.newActive("AgentGeneric", null);
          ItineraryServer iServer = (ItineraryServer) allItineraryServer.next();
          mobileAgent.setItineraryManager(new ItineraryManagerLocalNetwork());
          // nodes and SNMP
          mobileAgent.setLastDestination(rdvDestination);
          mobileAgent.getItineraryManager().prepareItinerary(iServer);
          mobileAgent.startItinerary();
        }
      }
     } catch (Exception e) {e.printStackTrace(); }
  }
  public void onArrival() {
    super.onArrival();
    SystemResources sr = new SystemResources();
    nodeRessource.add(nodeDest);
    nodeRessource.add(sr.getSystemOS());
    // Gets TOTAL MEM, USED MEM, FREE MEM into a String
    nodeRessource.add(sr.toString());
  }
  public void snmpOnArrival() {
    super.snmpOnArrival();
    // gets the SNMP MIB Table ip.ipNetToMediaTable
    AllArplTable.addAll(new IpMacTable(snmpDest).getArpTable());
  }
  public void displayData() {
  System.out.println("Data collected : "+listAgentsData);
  }
}
```

FIG. 6.20 – Classe de l'AgentClone
- agent principal

Chapitre 7

Performances et évaluation des agents mobiles pour l'administration système et réseau

7.1 Introduction

De manière récurrente l'utilisation des agents mobiles pour l'administration système et réseau donne les performances comme critère de préférence des agents mobiles face au modèle Client/Serveur SNMP. En effet, cela se traduit par une consommation de débit réseau moindre ([85],[101]). L'intérêt de ce chapitre est de conforter cette idée, dans le cadre de notre plate-forme. Nous allons montrer que, en ayant automatisé la création et le suivi d'itinéraires dynamiques, certaines opérations de gestion et d'administration ne requiérant pas de délai d'exécution sont utilement réalisables par des agents mobiles. Par contre, nous montrerons aussi que des contraintes subsistent quant à l'utilisation des agents mobiles pour l'administration notamment dans le cadre des réseaux sans fils (WLAN). Ce n'est pas l'évaluation du temps de création d'un itinéraire dynamique que nous cherchons à évaluer finement dans ce chapitre, car nous avons pu constater que lorsque la topologie du ou des réseaux est connue, ce temps de création de l'itinéraire varie toujours de quelques millisecondes à une seconde. Par ailleurs, le temps de construction de la topologie est lui même déjà évalué dans la section 5.5.5.

Par contre, il s'agit dans ce chapitre d'également valider l'exploitation de notre plate-forme dans un contexte réel, requiérant des opérations d'administration.

A cette fin, quelques performances où le débit du réseau et où le nombre d'hôtes varient vont être décrites et évaluées. On cherche à démontrer qu'il existe un seuil à partir duquel il devient plus intéressant d'envoyer un agent mobile pour réaliser la même fonction d'administration, car transmettre chaque variable

SNMP individuellement en Client/Serveur vers la station d'administration n'est pas rentable (en terme uniquement de performances). Il s'agit aussi de mettre en évidence le fait que le choix du Client/Serveur SNMP ou des agents mobiles n'est finalement pas qu'une question de performances, mais que ce choix dépend en fait foncièrement du type de fonction d'administration à réaliser, ou du type d'environnement physique (ce dernier pouvant quand même avoir une incidence sur des performances du réseau sous-jacent).

7.2 Analyse des performances

On suppose, dans l'évaluation de nos performances, que la plate-forme est déjà déployée sur tous les éléments intervenant dans la phase de test.

7.2.1 Expériences sur un réseau local

Nous avons cherché à évaluer, dans cette section, le temps d'exécution d'une opération d'administration à partir d'une station d'administration connectée sur un réseau distant.

7.2.1.1 Descriptif du réseau de test

Pour notre jeu de tests on se base sur deux réseaux (cf. figure 7.1). Sur le réseau A, la station d'administration (`Yaté`)et sur le réseau B, 11 ordinateurs de différentes puissances et capacités, dont la machine `Bourail`. Ces machines sont des PCs (fonctionnant sous Win95, Linux, WinNT, Win2K, WinNt Terminal Server) et des imprimantes en réseau (HP 4100 et HP 2100) avec un agent SNMP actif. Toutes ces machines seront la cible d'une opération d'administration.

Pour faire varier le débit du lien inter-réseau, nous avons utilisé un Pentium à 133Mhz (`Bourail`) fonctionnant sous FreeBSD avec `ip_dummynet` [74] (cf. figure 7.1) [1]. Par ce biais, nous pouvons simuler le changement du débit du lien inter-réseau de 100Kbps à 10Mbps. La bande passante de chaque sous-réseau (A et B) est de 100Mbps.

7.2.1.2 Itinéraire d'administration

L'itinéraire des agents mobiles consiste à effectuer une migration au départ pour aller de la machine `Yaté` vers la machine `Koumac`, une opération SNMP depuis `Koumac` vers chacun des éléments du réseau B, et une migration à chaque fin de l'itinéraire pour retourner sur le nœud ProActive de la machine `Yaté` afin de rapatrier les résultats de l'évaluation. L'itinéraire Client/Serveur consiste à

[1] Ce PC routeur héberge un agent SNMP actif du côté du réseau B donc nous le considérons comme un hôte administrable du réseau B

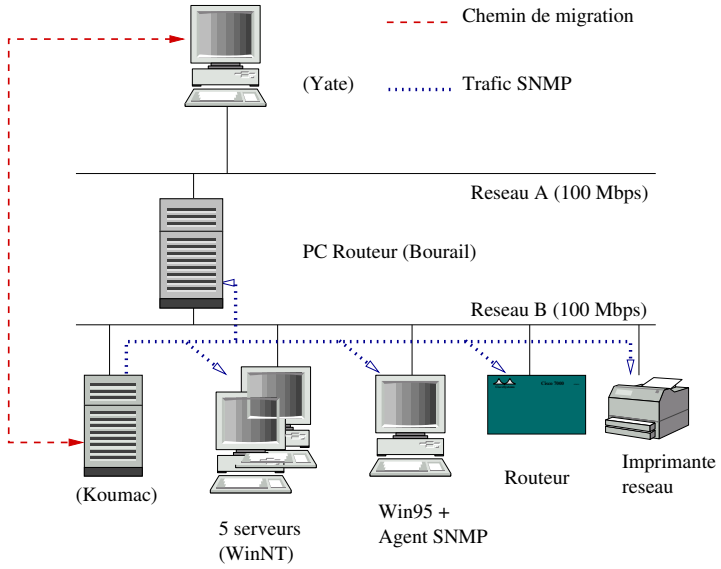

FIG. 7.1 – Schéma du réseau d'évaluation avec un débit modifiable sur le lien inter-réseau (via la machine bourail)

effectuer toutes les opérations SNMP depuis la machine `Yaté` vers chacun des éléments du réseau B. Dans les deux cas, pour simuler un réseau d'une taille plus importante, nous réitérons plusieurs fois la visite de chacun des éléments du réseau B (comme si sur le réseau B, il n'y avait pas 9 éléments mais bien plus).

7.2.1.3 Les opérations d'administration mises en œuvre

Pour évaluer les performances, nous avons mis en œuvre trois fonctions d'administration distinctes dans le but de faire varier le nombre de requêtes SNMP à l'intention de chaque agent SNMP. Nous avons utilisé la collecte du groupe *System*, de la table *IpRouteTable* et pour finir de la table *IpNetToMediaTable*, plus communément connue sous le nom de table ARP. Toutes les requêtes qui sont réalisées dans les jeux de test sont effectuées en mode séquentiel afin d'obtenir des écarts voulus dans les résultats.

L'objectif de cette évaluation est de pouvoir analyser le temps d'exécution des différentes méthodes en comparant la solution traditionnelle Client/Serveur et les agents mobiles pour l'administration. Nous faisons donc varier le débit inter-réseau géré par la machine `Bourail` afin de simuler des connexions distantes. Nous n'engendrons pas de perte de paquets, même si l'outil utilisé, `ip_dummynet`, le permet. En même temps, nous augmentons artificiellement le nombre d'hôtes pour essayer de tirer des conclusions permettant de donner la préférence à la solution Client/Serveur ou agent mobile selon les cas et les configurations de réseau testés.

Groupe System : La collecte des variables SNMP du groupe `System` (taille des paquets en moyenne pour cette fonction est de 42 octets [2]) sur un réseau ayant un lien faible débit, variant de 100Kbps à 200Kbps (cf. figure 7.2), permet de mettre en évidence que sur un nombre important d'agents SNMP le temps d'exécution total de l'opération d'administration en utilisant un agent mobile est meilleure. Alors que pour une quinzaine d'hôtes sur lesquels l'information est collectée, le temps d'exécution est comparativement le même, un écart sans cesse croissant apparaît au delà. On constate en fait, que la faible valeur du débit (de 100Kbps à 200Kbps) ne perturbe pas l'agent mobile puisque celui exécute la fonction d'administration en un temps total quasi similaire, alors même qu'il doit revenir à la station d'administration en transportant les données collectées.

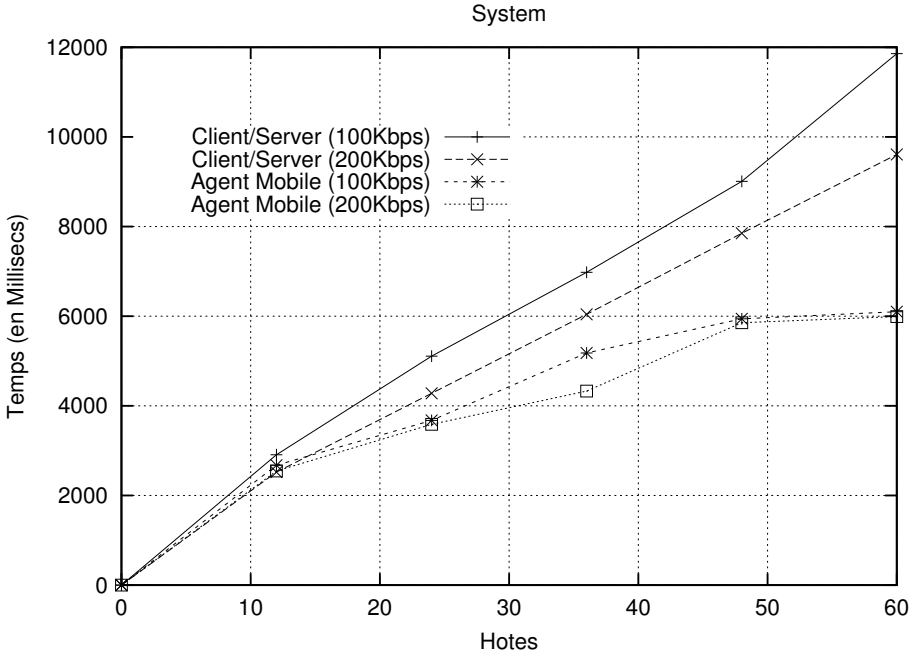

FIG. 7.2 – Collecte du groupe System (100Kbps-200Kbps)

Le même comportement est obtenu en augmentant le débit inter-réseau (de 1Mbps à 5Mbps) (figure 7.3). En conclusion de ces 4 configurations de test, dans lesquels le débit inter-réseau est bien moindre que le débit de chaque réseau, la technologie à agent mobile permet de tirer partie des débits réseau (100Mbps, bien mieux que le modèle Client/Serveur qui subit la lenteur du lien inter-réseau pour chaque lecture de variables SNMP.

Groupe IP-ipRouteTable : Afin d'expérimenter la collecte d'un nombre de variables SNMP non défini statiquement, nous avons collecté sur chaque agent

[2]uniquement pour la partie protocolaire SNMP

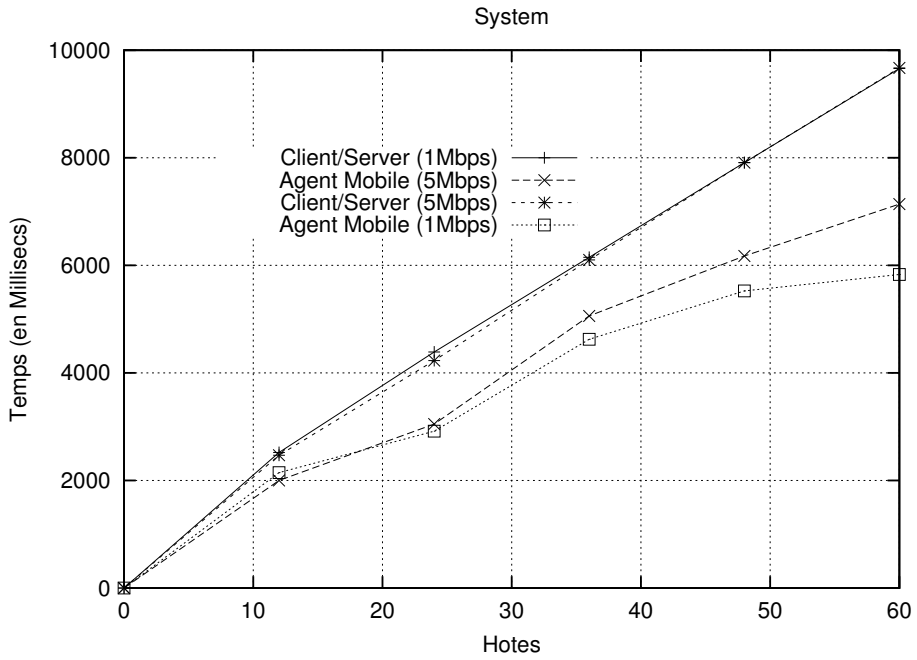

FIG. 7.3 – Collecte du groupe System (1Mbps-5Mbps)

SNMP la table de routage (taille des paquets en moyenne pour cette fonction est de 298 octets [3]). Le nombre minimal de variables qui sont collectées est de 6 (*ipRouteDest*, *ipRouteIfIndex*, *ipRouteNextHop*, *ipRouteType*, *ipRouteProto*, *ipRouteMask*) (, mais comme il s'agit d'une table et que plusieurs entrées dans celle-ci peuvent exister, il est impossible de déterminer à l'avance le nombre de variables collectées. Cette expérience est mise en œuvre sur les mêmes configurations de test que précédemment (cf. figure 7.4 et figure 7.5).

Par la quantité de variables collectées depuis la station d'administration, le modèle Client/Serveur est très pénalisé par le goulot d'étranglement induit par le faible débit du lien inter-réseau. Une nette différence se ressent par rapport à l'agent mobile qui lui, comme dans l'expérience précédente, tire un grand avantage du débit du réseau B à 100Mbps.

Groupe IP-ipNetToMediaTable : Il s'agit, ici aussi, de collecter un nombre important et non statiquement défini de variables SNMP et ce en allant lire les tables `ARP` des agents SNMP, tables qui contiennent en général plus de lignes que celles des tables de routage (taille des paquets en moyenne pour cette fonction est de 128 octets [4]). Il s'agit donc d'augmenter la quantité d'information que doit transporter l'agent mobile. Les variables que nous collectons sur chaque agent

[3] uniquement pour la partie protocolaire SNMP
[4] uniquement pour la partie protocolaire SNMP

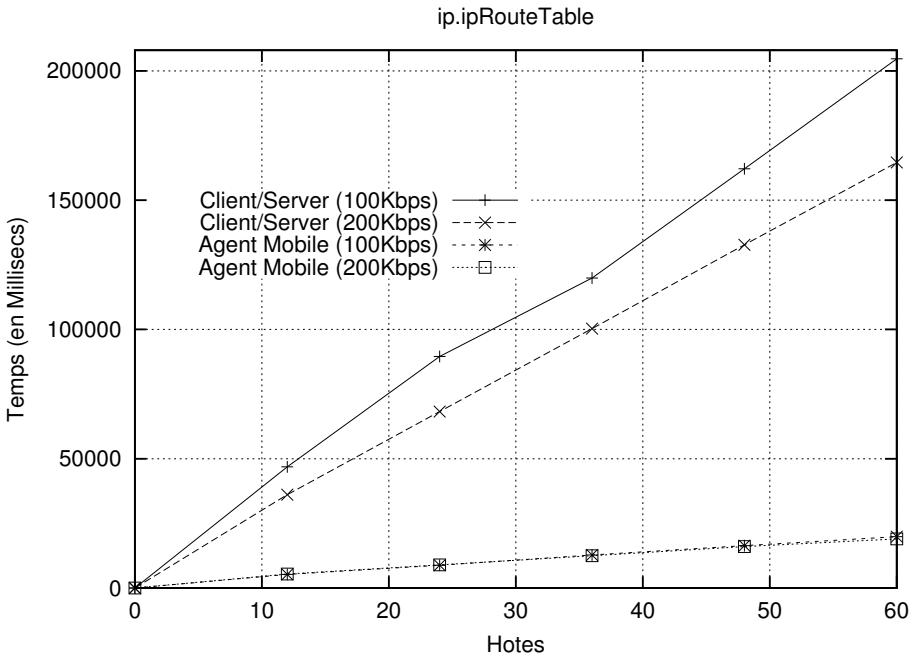

FIG. 7.4 – Collecte de la table de Routage (100Kbps-200Kbps)

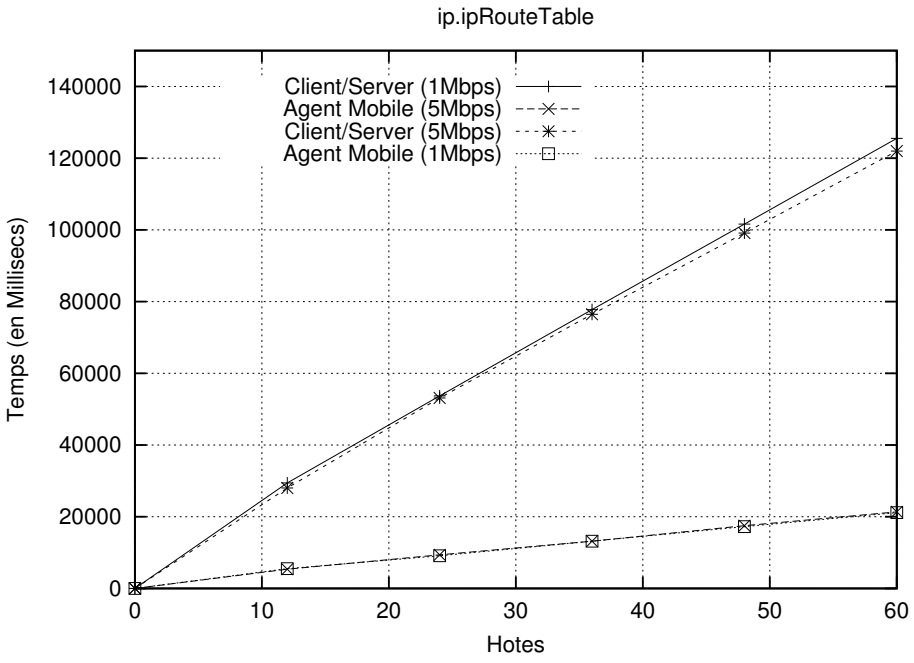

FIG. 7.5 – Collecte de la table de Routage (1Mbps-5Mbps)

SNMP sont dans la table *ipNetToMediaTable* (*ipNetToMediaIfIndex*, *ipNetToMediaPhysAddress*, *ipNetToMediaNetAddress*, *ipNetToMediaNetType*).

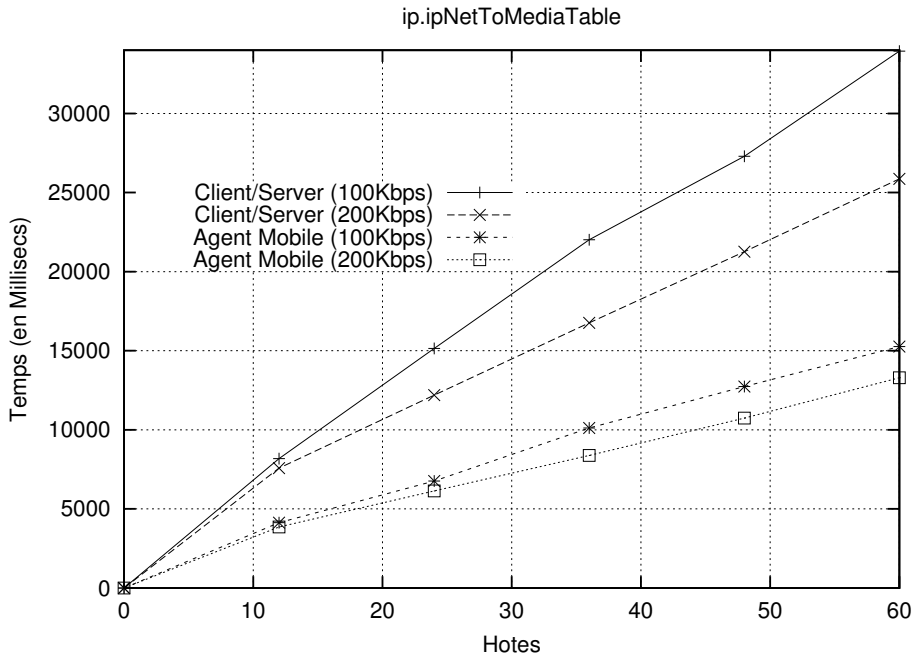

FIG. 7.6 – Collecte de la table ARP (100Kbps-200Kbps)

Les résultats obtenus et présentés dans le paragraphe précédent sont confortés lors de cette évaluation (figure 7.6 et 7.7). Plus l'information globale à collecter, à distance, est importante plus la technique à agent mobile devient avantageuse.

Comparatif des fonctions : Pour obtenir une vue plus exhaustive de l'ensemble de nos évaluations, nous regroupons les performances des fonctions d'administration collectant le groupe *System* et la table ARP (*ipNetToMediaTable*) et ce pour les différentes configurations où le débit du lien inter-réseau varie entre 10Kbps et 1Mbps (cf. figure 7.8). En considérant 12 éléments visités (avec agent SNMP), les meilleures performances sont celles obtenues par l'agent mobile qui effectue dans des délais plus courts les mêmes opérations que celles effectuées avec le modèle Client/Serveur traditionnel. On peut constater toutefois, sur la figure 7.8, que pour un nombre limité de variables SNMP (groupe `System`) à collecter et que pour un nombre d'éléments faible (ici 12 éléments), les deux solutions (agent mobile et Client/Serveur) sont presque identiques[5]. Un gain de temps substantiel est obtenu pour la fonction collectant la table ARP de chaque agent SNMP visité.

En conclusion, le modèle à agent mobile est très adapté pour effectuer des

[5]Certains fléchissements des courbes que nous avons obtenues sont le fait d'une expérimentation réalisée sur un réseau de production (réseau B). Nous avons choisi ce type d'expérimentation car celle-ci reflète une situation réaliste.

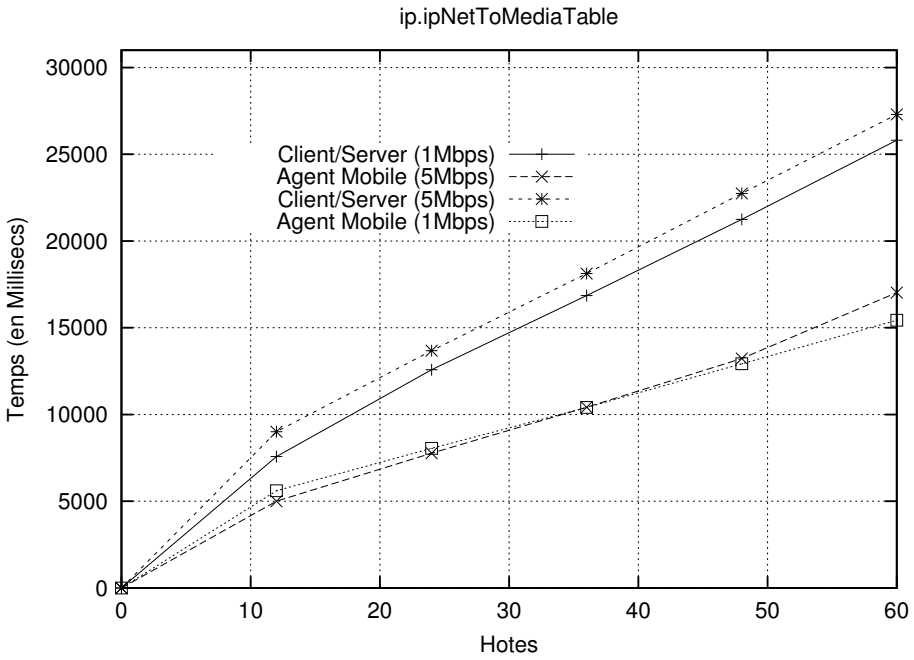

FIG. 7.7 – Collecte de la table ARP (1Mbps-5Mbps)

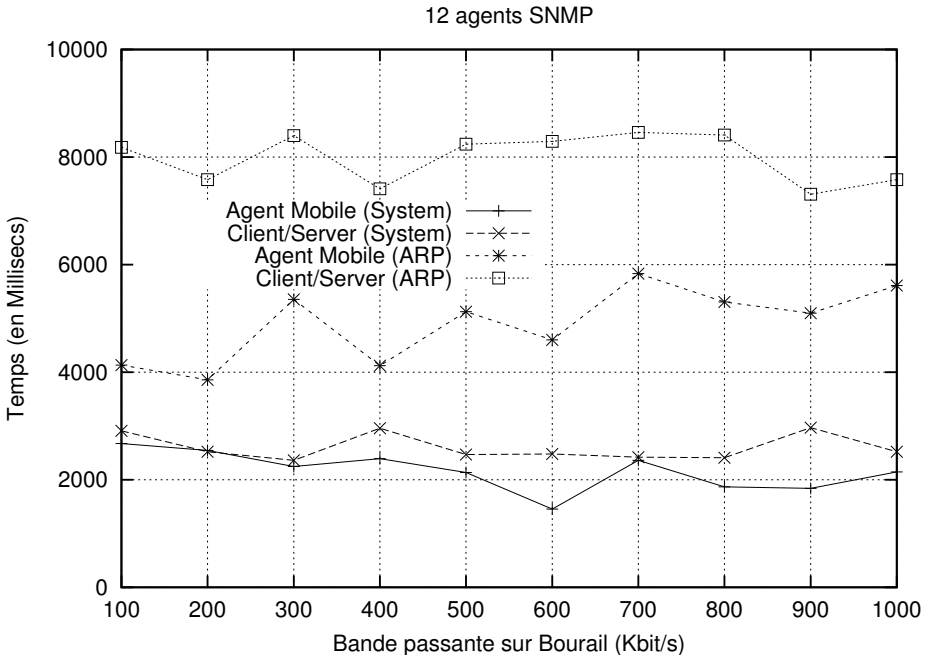

FIG. 7.8 – Comparatif des fonctions par débits

tâches de longue durée sur des réseaux distants, limitant ainsi l'utilisation de la bande passante inter-réseau. L'approche que nous avons utilisée dans ces expéri-

mentations est une approche de délégation de la tâche d'administration réseau, qui est caractérisée dans la littérature par le modèle *Static Delegated Model* [85]. Dans ce modèle, on tire un avantage de la position de l'agent mobile sur le réseau pour effectuer les opérations d'administration à distance, en lieu et en place de la station d'administration centralisée du modèle Client/Serveur (caractérisé par le *Static Centralized Model*). Nos expérimentations rejoignent ainsi les expérimentations et les conclusions obtenues par Simões [85].

7.2.2 Expériences sur un réseau WLAN

Nous avons cherché à évaluer le comportement des agents mobiles sur un réseau de nouvelle génération. Pour cela, nous avons pris en compte les réseaux de type hertzien et essayé d'obtenir des résultats de performance à comparer ensuite avec ceux obtenus lors des expérimentations précédentes.

7.2.2.1 Descriptif du réseau de test

Le jeu de tests que nous avons effectué est basé sur un réseau sans fil (technologie Wifi [59]) (cf. figure 7.9). Nous avons mis en œuvre ce test pour mettre en évidence les avantages et les inconvénients de la technologie des agents mobiles dans l'utilisation de ce type de réseau.

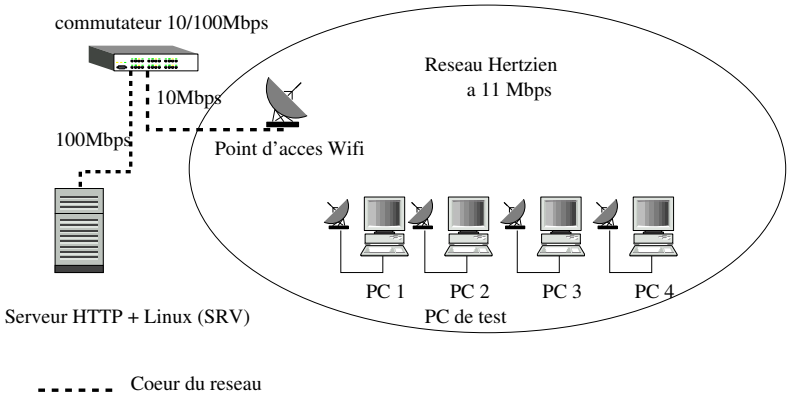

FIG. 7.9 – Schéma de la configuration d'évaluation avec un réseau sans fil

Le serveur sous Linux (SRV, RedHat 8.0) fonctionne sur un Pentium IV, 2.4 GHz, 512 Mo de RAM. Il est connecté au cœur du réseau par un lien à 100Mbps (commutateur Catalyst 2950). Le point d'accès Wifi fonctionne en mode Infrastructure (cf annexe pour plus de détails) et il est connecté en 10 Mbps sur ce même commutateur. Ce dernier dessert en mode hertzien à 11Mbps, 4 PCs. Les PCs (PC1, PC2, PC3 et PC4 dans les libellés des figures ci-dessous) sont des ordinateurs identiques basés sur des Pentium II cadencés à 300 MHz. Ils sont équipés de 64 Mo de RAM et ils fonctionnent sous WIN98. Chacun des

ordinateurs est relié au réseau par un adaptateur PCI Wifi. Un serveur HTTP, fonctionnant sous Apache, est hébergé par le serveur SRV. Ce service HTTP met à la disposition de la plate-forme à agents mobiles toutes les classes Java nécessaires au déploiement du code et des fonctions d'administration pour les agents mobiles (fonction système ou réseau).

7.2.2.2 Itinéraire d'administration

L'itinéraire de migration des agents mobiles dans le cas des tests présentés sur le réseau Wifi est le suivant :

- Départ de SRV impliquant une migration vers le PC1 connecté via une liaison sans fil,
- L'agent mobile migre d'un PC à l'autre jusqu'au dernier pour y effectuer l'opération d'administration,
- Après son passage sur le dernier PC (PC4), il migre vers SRV pour y effectuer la dernière opération d'administration et terminer son exécution.

L'itinéraire Client/Serveur consiste à effectuer l'opération d'administration sur tous les éléments du réseau (SRV, PCs) depuis SRV.

7.2.2.3 Les opérations d'administration mises en œuvre

Plusieurs séries de tests ont été réalisées avec la même fonction d'administration (le même code Java), que ce soit pour l'exécution en suivant le modèle Client/Serveur ou en utilisant un agent mobile. Toutes les requêtes qui sont réalisées dans les jeux de test sont effectuées en mode séquentiel, c'est-à-dire sans burst, en utilisant le même code de programmation.

- Les premiers tests réalisés sont basés sur le modèle Client/Serveur et servent de base pour une comparaison ultérieure.
- La deuxième série de tests utilise un agent mobile pour effectuer la fonction d'administration. On mesure les performances lors du 1er suivi de l'itinéraire défini pour ce réseau par l'agent mobile (les classes Java nécessaires à l'exécution de l'opération ne sont donc pas chargées).
- La troisième série de tests consiste à évaluer les performances lorsque le code de l'agent mobile et celui des fonctions d'administration ont déjà été chargés dans la JVM de chacun des éléments intervenant dans l'itinéraire. Nous cherchons à mettre en évidence l'intérêt du pré-chargement du code sur les nœuds de la plate-forme
- La quatrième série de tests permet de mettre en évidence le fait que le poids de l'agent mobile est pénalisant sur ce type de réseau. On mesure donc les améliorations possibles suite à la dépose des données transportées pendant les différentes étapes de migration.

Groupe IP-ipNetToMediaTable : Notre fonction de collecte de la table ARP (taille des paquets en moyenne pour cette fonction est de 128 octets [6])est tout d'abord exécutée en mode Client/Serveur à partir du serveur SRV. Le résultat de cette expérimentation (figure 7.10) donne un temps de collecte d'environ 4,5 secondes. Sur la même figure, on constate un temps d'exécution de cette fonction par l'agent mobile dix fois supérieur. Ce temps d'exécution est proportionnel au temps de chargement des classes Java d'administration et de l'agent mobile sur chacun des nœuds, et au fonctionnement général du réseau Wifi. Des performances plus intéressantes sont obtenues lorsque les classes Java sont déjà pré-chargées. Malgré cela, le temps d'exécution reste deux fois plus important que le modèle Client/Serveur.

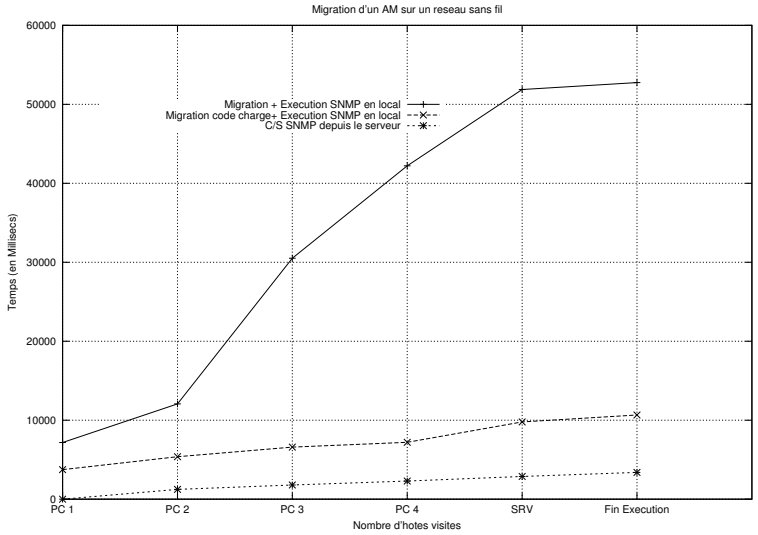

FIG. 7.10 – Interrogation des agents SNMP en Client/Serveur et avec agent mobile

Retour à la station d'administration : Dans ce jeu de test, l'ensemble des PCs à administrer est le même que précédemment, mais après chaque visite d'un PC, l'agent mobile revient sur la station de départ (SRV) pour déposer et traiter les données (cf. figure 7.11). Il est évident que ce type d'itinéraire (en "étoile") n'est pas réaliste sur ce type de réseau à faible débit, à cause du coût engendré par de nombreuses migrations.

Toutefois, un modèle d'administration plus pertinent consiste à utiliser des agents stationnaires sur chaque nœud du réseau Wifi et les interroger à distance, par exemple, à partir du serveur d'administration. Ces agents stationnaires peuvent être simplement créés à partir d'un agent mobile d'administration et déployés sur chacun des nœuds du réseau. Dans une telle configuration, le réseau

[6]uniquement pour la partie protocolaire SNMP

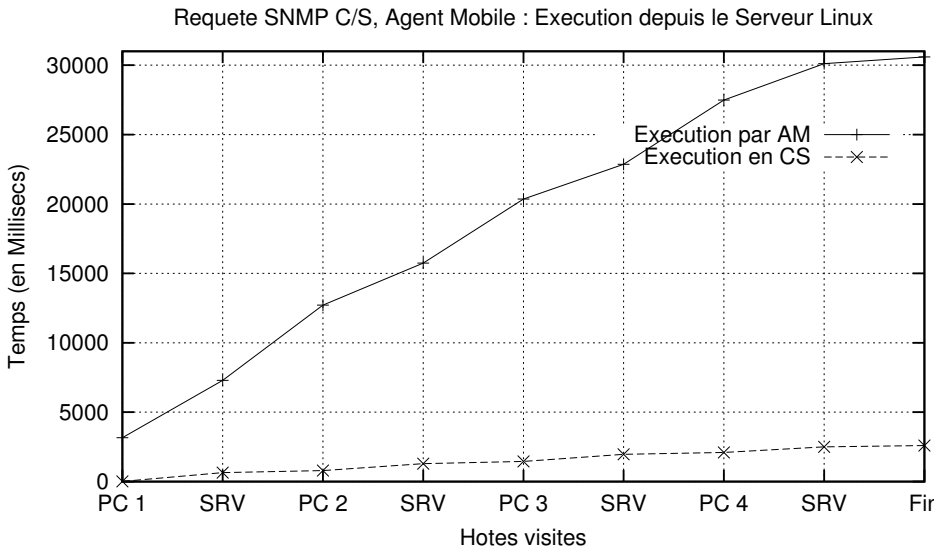

FIG. 7.11 – Retour systématique à la station de départ

ne sert que pour la collecte des données que chaque agent stationnaire a préparé pour l'agent d'administration maître localisé sur la station d'administration.

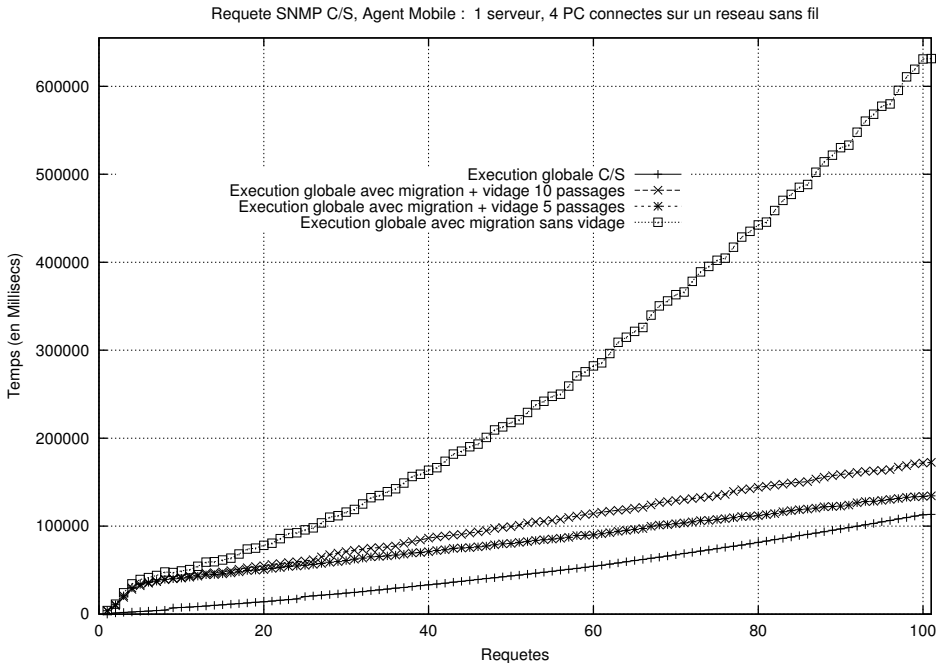

FIG. 7.12 – Estimation du vidage de l'agent mobile

Optimisation : Constatant que faire migrer un agent mobile sur un tel réseau est très pénalisant, d'autant plus si la taille de l'agent mobile est importante, on a pensé qu'il serait judicieux d'intercaler dans l'itinéraire d'administration le fait d'aller de temps en temps seulement, sur un nœud (SRV) dans le but d'y déposer ses données pour "maigrir". On a mesuré les gains obtenus, en variant la fréquence de dépôt (cf. figure 7.12). Malgré les migrations supplémentaires imposées pour aller "maigrir" à des emplacements prévus, on constate qu'il y a un gain réel.

7.2.2.4 Itinéraire de migration en mode ad hoc

Il existe une contrainte de localisation des nœuds dans un réseau Wifi en mode ad hoc (cf. annexe Wifi). Un nœud n'a pas forcément la visibilité de tous les autres nœuds, ce qui nécessite d'avoir dans notre cas des itinéraires d'administration qui doivent être modifiés à la volée (cf. figure 7.13).

Par exemple, prenons l'itinéraire de migration que nous avons utilisé lors de nos expérimentations. SRV→PC1, P1→PC2, PC2→PC3, PC3→PC4 et PC4→SRV. Si dans le réseau Wifi en mode ad hoc, les zones de couvertures hertziennes sont les suivantes :

- PC1 et PC3 sont en zone de couverture
- PC3 et PC2 sont en zone de couverture
- PC2 et PC3 sont en zone de couverture
- Le point d'accès est visible par PC1 et PC4

Cela veut dire que PC1 n'est pas visible de PC2 et vice versa, et que le système Wifi transmette donc les données via un nœud intermédiaire (par exemple PC3). Pour éviter de passer par des nœuds intermédiaires ce qui augmente le délai d'exécution, on peut par exemple préparer un itinéraire d'administration dans lequel toute paire d'éléments successifs est formée d'éléments directement accessibles. Par exemple : SRV→PC1, PC1→PC3, PC3→PC2, PC2→PC4 et pour finir PC4→SRV.

Une telle idée a été proposée par Migas [55] en utilisant des agents stationnaires pour essayer de déterminer la politique de migration la plus adaptée pour les agents mobiles dans un réseau Wifi de type `ad hoc`.

7.3 Exemples d'utilisation des agents mobiles

7.3.1 Quelques exemples simples exploitant la propriété d'autonomie

En constatant que les réseaux actuels offrent de plus en plus de bande passante (LAN, WAN, WLAN), nous allons introduire des exemples concrets d'utilisation des agents mobiles dans un cadre d'administration système et réseau. Ces exemples permettent de justifier le choix de la technologie à agent mobile en

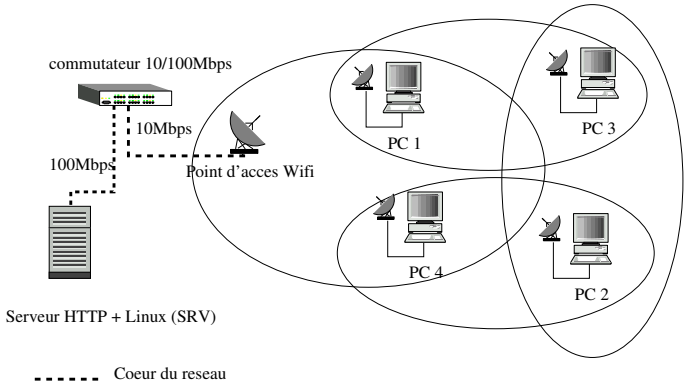

FIG. 7.13 – Schéma d'un réseau Wifi fonctionnant en mode ad hoc

fonction, non pas des performances et du gain en bande passante escompté, mais en fonction du service rendu. Des critères tels la réactivité, l'autonomie, la prise en compte de contraintes géographiques sont alors considérés.

Imaginons par exemple, qu'un administrateur veuille effectuer un inventaire automatique de tous les systèmes connectés au réseau de son entreprise. Pour cela, on met en œuvre des opérations Java et des requêtes SNMP afin de collecter le descriptif des systèmes, mais aussi sur chaque hôte (PC, serveur) ayant la MIB `host` implanté, toutes les ressources du système. Dans cet exemple, le temps de réponse n'est pas critique puisque la tâche est une tâche de fond, de longue haleine. Il s'agit de déployer plusieurs agents mobiles respectant l'architecture du réseau, c'est-à-dire déployés sur l'ensemble des sites. Avec l'autonomie propre de chaque agent, lorsque l'opération d'administration est menée à terme, chaque agent mobile peut ramener à la station d'administration les données collectées ou alors les transmettre à l'agent responsable de la mise en forme de celles-ci. Toutes ces opérations peuvent être effectuées indépendamment de la station d'administration.

Prenons un autre exemple : un service d'impression sur le réseau s'arrête de manière inopinée. En urgence, l'administrateur va envoyer un agent mobile dédié à la supervision de ce service afin de savoir rapidement la cause de cet arrêt. En même temps, l'agent mobile peut essayer de relancer le service défectueux, ou de rediriger toutes les impressions bloquées vers un autre service d'impression. Une solution plus autonome, consiste à utiliser des agents mobiles de supervision qui n'attendraient pas d'ordre direct de l'administrateur pour réagir à toute modification des services réseaux fournis.

7.3.2 Exemple : Autoconfiguration des VLANs dans un réseau

Imaginons un administrateur de réseau qui doit déployer une nouvelle architecture de réseau, bien que notre exemple puisse s'appliquer aisément dans le cas d'une architecture réseau en exploitation. L'administrateur du réseau prépare les configurations de base des équipements actifs du réseau, en fonction de ses desiderata : Nom logique, adresse Ip, nom de communauté SNMP V1 en lecture seule, informations sur la configuration SNMP V3 nécessaire pour les opérations d'administration, et ce pour chaque commutateur appartenant au cœur du réseau. Il définit aussi des réseaux virtuels (*VLANs*) afin de contrôler les accès entre les différents éléments qui seront connectés sur le réseau : PC, serveurs, imprimantes, dans l'optique de faire des groupes de machines par réseau virtuel. Pour réaliser cela, l'administrateur doit connaître le lien entre chacune des interfaces sur les commutateurs, les éléments qui doivent y être connectés et le VLAN dans lequel ces éléments doivent être intégrés. Par défaut, les configurations des commutateurs associent chaque port au VLAN administratif, qui est le VLAN de gestion du commutateur. Pour changer cette valeur par défaut, l'administrateur doit donc, soit se connecter sur le commutateur pour en changer la configuration (en utilisant un client ssh [48], par exemple), soit utiliser pour des raisons de sécurité et de distance, le protocole SNMP V3[7]. Dans tous les cas, l'administrateur devra connaître l'adresse IP du commutateur sur lequel il doit effectuer l'assignation du VLAN sur une interface de ce commutateur.

En nous basant sur l'exemple proposé par la figure 7.14, où le VLAN 180 est déjà déclaré pour certains éléments, nous allons expliciter comment nous utilisons un agent mobile pour configurer les VLANs sur les ports des commutateurs.

Ainsi, en utilisant un itinéraire de migration dans le réseau local, ou vers un réseau local distant, et en fournissant comme information à l'agent mobile : (1) l'adresse Ethernet de l'élément concerné par l'assignation du VLAN ; (2) le numéro du VLAN pour configurer l'interface du commutateur, l'agent mobile peut automatiquement configurer l'interface du bon commutateur de manière complètement automatique. En effet, notre algorithme de construction de la topologie nous permet de détecter automatiquement l'équipement actif sur lequel l'action de l'agent mobile doit être effectuée.

7.4 Bilan

Par les tests que nous avons mis en œuvre, sur le réseau présenté par la figure 7.1, nous pouvons constater que la technologie des agents mobiles donne

[7]attribut de l'entrée de la MIB : private.cisco.ciscoMgmt.ciscoVlanMembershipMib.ciscoVlanMembershipMIBObjects. vmMembership.vmMembershipTable.vmMembershipEntry.vmVlan

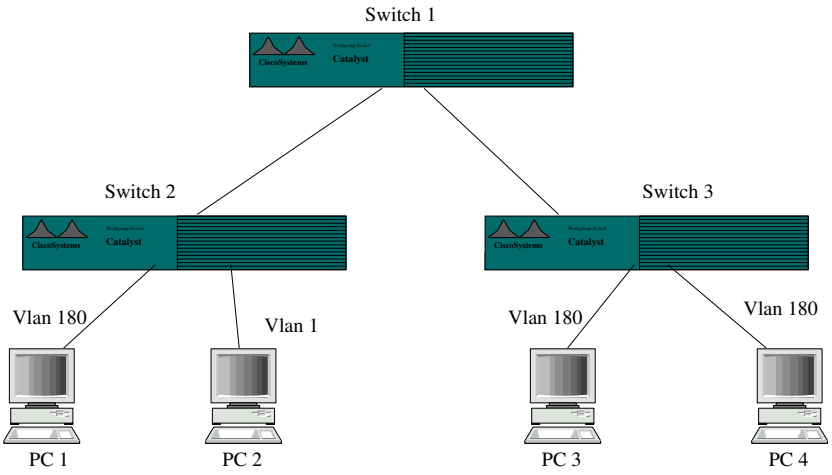

FIG. 7.14 – Schéma du réseau pour l'autoconfiguration des VLANs

des avantages en temps d'exécution et que les résultats obtenus ne sont pas trop dépendants du débit inter-réseau. A contrario, les tests montrent effectivement que le modèle Client/Serveur exécuté à distance n'est pas adapté à la collecte d'un nombre important de variables SNMP sur plusieurs agents SNMP. La mise en œuvre de tâches longues qu'un administrateur souhaite voir exécutées est tout à fait envisageable en utilisant des agents mobiles, notamment lorsque les éléments sont situés sur un réseau distant et qu'il est possible de réaliser des opérations sur des éléments qui n'étaient pas concernés au départ grâce aux itinéraires construits dynamiquement. Il convient de constater que lorsque les fonctions sont à exécuter sur le réseau local, les deux techniques (Client/Serveur et agent mobile) sont aussi bien applicables l'une que l'autre.

Quand on travaille sur un réseau de nouvelle génération (type Wifi), il est nécessaire de prendre en compte les contraintes qui y sont associées : latence, faible bande passante, collision, mode de fonctionnement du réseau (infrastructure ou ad hoc). Ainsi nous avons pu constater par l'évaluation des performances d'agents mobiles agissant sur un réseau Wifi que des précautions doivent être prises : il faut alléger l'agent mobile pour une meilleure efficacité ; éviter les aller-retour inutiles pendant l'itinéraire, ce qui impose que l'itinéraire d'administration soit très adapté à ce type de réseau ; l'utilisation d'un agent mobile stationnaire permet sur ce type de réseau de regrouper et mettre à disposition des informations d'administration sans pour autant engorger le réseau Wifi.

Pour conclure ce chapitre, nous pensons que les agents mobiles trouvent une place dans le monde de l'administration système et réseau mais que cette technologie n'est pas adaptée à toutes les fonctions d'administration réseau (par exemple une fonction de courte durée sur un réseau local). Il est préférable d'utiliser les agents mobiles pour des tâches "plus longues", réalisables de manière décentralisée et autonome, et n'imposant pas une urgence particulière.

Chapitre 8

Conclusion

8.1 Bilan, contribution

Nous nous sommes intéressés dans cette thèse à la problématique de l'utilisation des agents mobiles dans un cadre d'administration système et réseau, en utilisant la bibliothèque ProActive, et au développement d'une plate-forme à agents mobiles pour l'administration système et réseau.

Nous avons étudié les plates-formes qui relèvent de cette problématique (par exemple, James, SOMA, Ajanta, etc..) afin de cerner la problématique sur l'automatisation et l'autonomie des agents mobiles. Nous avons constaté que des itinéraires statiques étaient fournis aux agents mobiles ce qui a motivé la définition d'itinéraires plus dynamiques et mieux adaptés aux opérations d'administration que les agents mobiles doivent accomplir.

Pour créer des itinéraires dynamiques, la connaissance de la topologie du réseau s'est avérée nécessaire. Il a fallu collecter l'information contenue dans les équipements actifs du réseau, et déterminer les services que ces éléments fournissent au réseau. L'information collectée permet d'obtenir des listes d'éléments représentant les équipements actifs et les nœuds ProActive. De même, nous utilisons cette information pour construire une topologie réseau de niveau 2. Cette information alimente au fur et à mesure des serveurs sur le réseau (ItineraryServers).

En étendant le modèle des `Destinations` de ProActive, nous avons pu définir les types d'éléments à administrer en fonction des services que ces éléments mettent à disposition des autres systèmes du réseau : `NodeDestination` pour les opérations d'administration de systèmes (impliquant une migration préalable des agents mobiles) et `SNMPDestination` pour l'administration de réseau (en Client/Serveur SNMP). L'architecture proposée permet de rajouter d'autres types de `Destinations` en fonction des besoins particuliers de l'administrateur. Pour utiliser, de manière transparente et automatique ces `Destinations`, nous avons mis en œuvre un gestionnaire générique d'itinéraire, l'`ItineraryManager`

qu'il suffit de spécialiser selon le type d'itinéraire voulu. Ce gestionnaire permet de construire des itinéraires adaptés aux opérations d'administration, et de les faire suivre par des agents mobiles d'administration. Nous avons pu, par exemple, proposer des itinéraires fonctionnant par sous-réseau dans le domaine administré : pour des équipements actifs, pour des systèmes ayant des nœuds ProActive, et ou toute combinaison des deux. Pour étendre notre solution sur l'ensemble du réseau, nos itinéraires se construisent à la volée, au moment où l'agent mobile doit changer de sous-réseau. En se basant sur ce mécanisme d'itinéraires, nous avons défini un cadre de programmation d'agents mobiles d'administration.

Nous avons pu montrer, par les évaluations de performances que nous avons réalisées, que l'utilisation des agents mobiles dans un cadre d'administration système et réseau se justifie, mais sous certaines conditions. De manière résumée, l'utilisation des agents mobiles sur le même sous-réseau que celui où se situe la station d'administration n'est pas déterminant en terme de performance. En effet, il existe un *overhead* lié à la plate-forme Java par rapport à des requêtes SNMP simples (par exemple, collecte de la table ARP). Dès qu'il s'agit d'administrer des éléments sur des sous-réseaux distants reliés par un lien dont le débit est faible, les performances des agents mobiles sont bien meilleures que celles du Client/Serveur. Par ailleurs, un certain avantage est obtenu pour des fonctions d'administration de "longue durée", qui peuvent bénéficier de l'autonomie des agents mobiles. Pour des fonctions d'administration plus simples, le choix de la technique (agent mobile ou Client/Serveur) dépend de l'administrateur.

Nous avons de plus, testé les environnements réseaux hertziens (*Wifi*) et constaté que les agents mobiles stationnaires constituent une solution mieux adaptée que du SNMP Client/Serveur classique. Toutefois, les agents mobiles peuvent être utilisés dans cet environnement mais leurs itinéraires d'administration doivent être adaptés au mode opératoire du réseau sans fil (cf. annexe 9.3) : mode infrastructure ou ad hoc.

La souplesse de ProActive et de notre plate-forme d'administration système et réseau permet d'envisager la mise en œuvre de tous les modes d'administration répertoriés [85, 54] tels que :

- `Migratory Model :` un seul agent migre sur l'ensemble des nœuds du réseau (notre type d'itinéraire est l'`ItineraryManagerWholeNetwork`)
- `Master/Worker Model :` le modèle est basé sur une mobilité réduite et une grande distribution des agents. En effet, la station d'administration déploie un agent par nœud à superviser et les interroge à la demande. Dans cette approche en parallèle des tâches d'administration, les performances globales du système d'administration sont améliorées car toutes les tâches d'administration sont effectuées localement sur les nœuds. (notre type d'itinéraire adapté est l'`ItineraryManagerClone`)
- `Static Delegated Model, Mobile Distributed Manager :`

une hiérarchisation des agents mobiles par domaine, permet à la station d'administration d'être en contact uniquement avec les agents mobiles maîtres, qui sont en charge de préparer les données des opérations d'administration. A leur tour, les agents mobiles maîtres interrogent des agents mobiles secondaires qui effectuent les opérations d'administration, pour collecter les données recueillies. (notre type d'itinéraire adapté est l'`ItineraryManagerLocalNetwork` associé à la description du sous-réseau cible)

– `Migratory Delegated Model` : le modèle est basé sur une répartition intelligente des agents mobiles sur l'ensemble du domaine à administrer en utilisant la localisation "de proximité" plutôt que les multiplications des agents mobiles. En effet, cette approche propose un agent mobile par sous-réseau qui supervise tous les nœuds du sous-réseau (notre type d'itinéraire est l'`ItineraryManagerLocalNetwork`)

En résumé, nous avons proposé une plate-forme d'administration système et réseau basée sur des agents mobiles qui suivent automatiquement des itinéraires dynamiques et construits à la volée. Cette plate-forme d'administration système et réseau offre un cadre évolutif de programmation permettant d'adjoindre facilement de nouvelles fonctionnalités dans la palette de fonctions offerte habituellement à un administrateur.

8.2 Administration dans le contexte des applications

On peut souligner le fait que l'utilisation des agents mobiles dans un cadre d'administration de systèmes et de réseaux (au sens classique du métier d'administrateur système et réseau) peut s'avérer parfois trop coûteuse en temps d'installation et de mise en œuvre, surtout pour un administrateur de réseau qui ne serait pas au fait de la programmation et du déploiement d'applications en environnement réparti (un cadre d'administration tel celui que nous avons développé constitue bien une application répartie et non triviale à déployer et à utiliser).

En revanche, une fois une telle difficulté maîtrisée, cette approche permet de tirer partie de la bonne connaissance du réseau et de fournir des techniques de gestion permettant d'automatiser la supervision d'un vaste ensemble hétérogène de machines et d'équipements, tel qu'illustré, par exemple, par des grilles de calcul. Par exemple, obtenir l'état (disponibilité) de tous les nœuds d'une grille de calcul, peut être entièrement délégué à un ensemble d'agents mobiles qui renseigneraient soit un ensemble d'administrateurs de la grille, soit des agents de calcul, sur l'état de la grille. De tels agents jouent alors le rôle de sonde (c'est pour cela qu'on les rend souvent stationnaires). En considérant qu'un tel système de supervision est mis à la disposition d'agents de calculs dans un premier temps, le système qui

les déploie s'adresse aux agents sondes et cela lui permet ensuite de décider de la machine d'accueil. Dans un deuxième temps, si les calculs sont mobiles on peut imaginer les agents qui les exécutent capables de migrer afin de mieux utiliser les ressources d'autres systèmes. En effet, un problème classique mais réel en réparti, est de décider où exécuter les calculs. Pour cela il faut avoir des moyens pour connaître la charge des systèmes et des réseaux, le nombre d'agents présents et le temps CPU qu'ils utilisent [32], le nombre de threads disponibles, etc.., tous les paramètres utiles à la prise de décision du placement des agents mobiles de calcul au moment de leur déploiement mais aussi pendant leur cycle de vie.

On voit bien dans cet exemple que les fonctionnalités de supervision (offertes notamment par NWS [103], un outil incontournable pour la grille) sont offertes à des couches applicatives (ici, le service de répartition de charge) qui, à l'instar d'un administrateur humain, ont à prendre des décisions.

De manière orthogonale, la complexité croissante des services et des applications déployés sur les infrastructures réparties justifie la transposition des techniques standard d'administration, des systèmes et des réseaux à ce nouveau contexte. Un outil comme JMX (voir annexe 9.2) constitue un bel exemple d'outil qui permet de favoriser cette transposition. En effet, JMX permet d'ouvrir les composants applicatifs au monde de l'administration et notamment, à ses standards comme SNMP. Supposons un ensemble de composants applicatifs administrables via JMX, il devient nécessaire de concevoir des outils intégrés, donc des plates-formes d'administration pour superviser ces éléments logiciels (telles que par exemple James [83] (plate-forme d'administration à agents mobiles, elle-même administrable via JMX), Lira [21], projet OLAN [58], SmartFrog [40]). Les soucis d'automatisation, de facilité de programmation des fonctions d'administration, de performance, notamment sur des réseaux à faible débit, etc..., nous semblent à nouveau justifier l'emploi d'agents mobiles dans ces plates-formes. Nous pensons donc que le travail décrit dans cette thèse trouve son application. Nous allons essayer à présent d'illustrer cette éventuelle application.

Étant donné une application complexe s'exécutant sur le réseau que l'on désire administrer, en utilisant des agents mobiles d'administration suivant des itinéraires dynamiques, nous proposons de considérer que les agents vont se déplacer, non pas sur un groupe d'éléments constituant un réseau, mais sur un groupe de composants de l'application constituant un groupe de services.

En effet, c'est pertinent de considérer que des applications et des services peuvent être regroupés par type de service fourni, et ce, même en se plaçant au niveau réseau. Par exemple, le service d'impression global d'un réseau est composé du service d'impression de chaque serveur d'impression et de chaque imprimante quelqu'en soit le type. Ainsi il est possible, en utilisant la technologie offerte par JMX, d'instrumenter chaque élément du groupe de service et des applications disponibles. Chaque application peut ainsi devenir administrable par un protocole tel SNMP en ayant au préalable défini la MIB SNMP associée.

Ensuite, il faut programmer les opérations d'administration de ces services au sein d'agents mobiles (ProActive), et ce, selon notre cadre de conception. Cela passe aussi par la définition de nouvelles `Destinations` qui permettront aux agents mobiles d'administration d'atteindre, peut être juste virtuellement, les agents JMX et d'y effectuer les opérations d'administration. A partir de telles `Destinations`, on pourra donc prévoir de nouveaux types d'itinéraires adaptés à ce type d'administration d'applications et les faire suivre à nos agents mobiles qui feront en sorte que les bonnes méthodes d'administration soient exécutées.

Cette technique appliquée unitairement sur chaque élément logiciel peut être utilisée à un niveau supérieur dans le sens où l'on voudrait pouvoir administrer un service indépendamment du nombre d'applications et de composants logiciels qui mettent en œuvre ce service. Par exemple, on peut considérer un groupe de services administrables par JMX que l'on aimerait superviser comme un tout, et ce de façon très simple, à partir d'un navigateur Web ou d'une plate-forme d'administration réseau standard. Pour cela, on associerait à un tel groupe les types d'agents mobiles capables d'administrer ces composants et leurs itinéraires. Ce même groupe de services serait à son tour rendu administrable à distance, soit en utilisant le protocole HTTP soit le protocole SNMP, grâce à l'instrumentation fournie par JMX. On pourrait dès lors déclencher les opérations d'administration associées à chaque groupe de services, sans avoir à en connaître le mode de fonctionnement.

Chapitre 9

Annexes

9.1 La technologie Jini

La technologie Jini [53] permet d'enregistrer des objets Java sérialisés dans un service d'enregistrement (cf. figure 9.1). Ce service d'enregistrement est à l'écoute sur le réseau multicast des différentes demandes provenant d'autres objets Java. Il peut exister plusieurs services d'enregistrement par sous-réseau. Chacun de ces services d'enregistrement répond indépendamment les uns des autres. Sur un domaine réseau, il est donc possible d'avoir un service d'enregistrement par sous-réseau du domaine d'administration pour améliorer la fiabilité du système d'enregistrement.

9.1.1 Les services enregistrés

Les services qui doivent être enregistrés, sont représentés par une instance de leur classe qui sera enregistrée dans le Lookup Service.

L'interface du service, qui est enregistrée dans le Lookup Service, peut donner accès au service de plusieurs façons :

- L'interface peut représenter le service entièrement, et dans ce cas, le service lui-même est transmis au client lors du processus de recherche et est ensuite exécuté en local par le client.
- L'interface peut aussi n'être qu'un recueil de déclarations de méthodes, ce qui fait que l'interface se comporte comme un *proxy*. Lorsque le client invoque une méthode, l'interface fait suivre la demande vers le fournisseur du service qui exécute l'instruction.
- L'interface représente une approche intermédiaire des deux précédentes. Elle consiste à faire exécuter une partie du service en local et une partie par le fournisseur. Les proxies qui utilisent cette approche sont appelés des *proxy* intelligents.

9.1.2 L'enregistrement d'un service

L'enregistrement d'un service dans le Lookup Service se déroule de la manière suivante (cf. figure 9.1) :

- Recherche du Lookup Service par le fournisseur de service
- Le Lookup Service retourne un objet, appelé le *Registrar*, qui agit comme un proxy du Lookup Service. Toute les requêtes à destination du Lookup Service se font via le *Registrar*
- Le fournisseur de service enregistre une copie de l'objet du service dans le Lookup Service

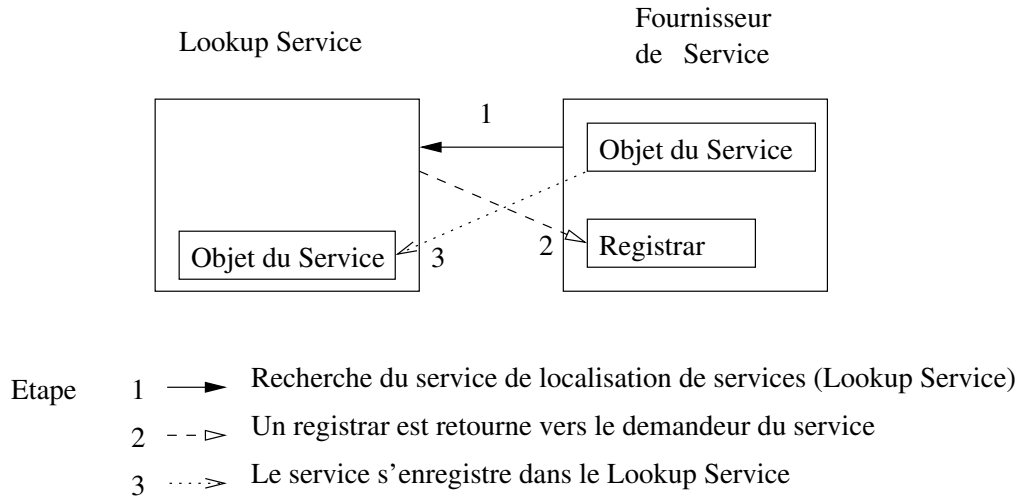

FIG. 9.1 – Jini du côté serveur

9.1.3 La localisation d'un service

Du côté du client (cf. figure 9.2), le client veut obtenir une copie du service dans sa propre JVM. Il utilise le même mécanisme que pour l'enregistrement d'un service (voir ci-dessus) pour récupérer un Registrar. Ensuite, il demande au Lookup Service une copie du service qu'il recherche.

9.1.4 Utilisation de Jini dans notre plate-forme

Nous utilisons donc les possibilités d'enregistrement et de localisation de Jini pour enregistrer la localisation de nos objets actifs et pour permettre aux agents mobiles de les contacter sans avoir à connaître leur localisation (par exemple l'URL de leur localisation,etc.).

Les objets de notre plate-forme d'administration système et réseau que nous enregistrons dans le Lookup Service sont les suivants :

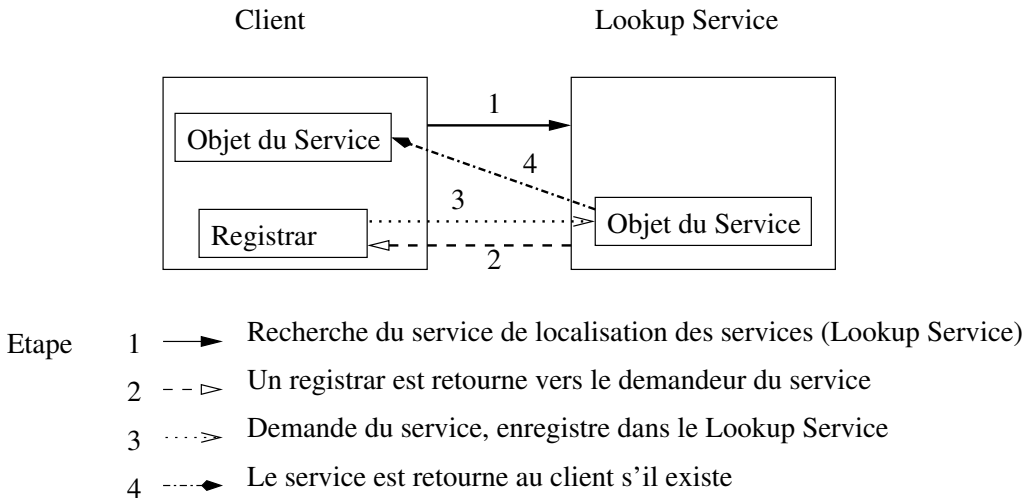

FIG. 9.2 – Jini du côté du client

- `ItineraryServerProxy` qui est une instance de la classe `ItineraryServer-Interface` implémentée par les `ItineraryServers`. Lorsque cet objet est récupéré via Jini, il est possible d'avoir la localisation (la référence du service) du ou des `ItineraryServers` qui s'exécutent.
- `NetworkDescription` qui est un objet encapsulant les descriptifs des sous-réseaux (`SubNetworkDescriptions`), avec des méthodes permettant l'extraction , par exemple, d'un `SubNetworkDescription` selon une adresse de réseau.

9.2 Java Management Extension

Les spécifications Java Management eXtensions (JMX) [52] décrivent un standard destiné à l'administration et la supervision d'applications Java à travers le réseau.

JMX définit une couche d'isolation entre les ressources à gérer (objets Java) et le système d'administration. Cette couche intermédiaire est basée sur un standard permettant de réaliser des "objets JMX" ou "MBeans" (managed beans) qui vivent dans un "conteneur JMX" et qui offrent une capacité d'administration centralisée à partir des différents outils. Un client JMX peut accéder aux attributs et invoquer des méthodes d'un "MBean" ; il peut également recevoir des notifications émises par un "MBean".

9.2.1 Les niveaux dans JMX

Instrumentation : Les ressources, comme une application, des éléments du réseau ou des services, sont instrumentés en utilisant des objets Java appelés

Managed Beans (MBeans). Un MBean expose ses interfaces de gestion, composées d'attributs et d'opérations, au travers d'un agent JMX pour une administration et une supervision à distance. Ainsi, n'importe quelle ressource qui doit être administrée en dehors de la JVM doit s'enregistrer en tant que MBean dans un serveur MBean.

Agent : Le composant principal d'un agent JMX est le serveur MBean. C'est un serveur d'objets administrés (le cœur) dans lequel les MBeans sont enregistrés. Un agent JMX incorpore aussi un ensemble de services pour gérer les MBeans. Les agents JMX contrôlent directement les ressources et les rendent disponibles pour des agents d'administration distants.

L'administration distante d'agents JMX : Les adaptateurs de protocoles et les connecteurs standards font qu'un agent JMX est accessible par des applications d'administration distantes vis à vis de la JVM qui héberge l'agent JMX.

Un adaptateur de protocole permet d'avoir une vue au travers d'un protocole spécifique (par exemple HTTP, SNMP ou propriétaire) de tous les MBeans qui sont enregistrés dans le server MBean.

Un connecteur fournit une interface du côté de l'application d'administration qui gère la communication entre l'application et l'agent JMX. Chaque connecteur fournit la même interface au travers des différents protocoles. Quand une application d'administration utilise cette interface, le connecteur peut la mettre en relation, de manière transparente, avec un agent JMX par le réseau, quelque soit le protocole utilisé.

9.3 Wifi

Il existe deux modes d'accès aux réseaux sans fil, le mode infrastructure et le mode ad hoc. Chacun des modes diffère par la manière utilisée par les éléments pour communiquer entre eux.

9.3.1 Les réseaux sans fil en mode Infrastructure

Les réseaux sans fil avec "Point d'accès" (ou "access point", AP en anglais) sont des réseaux qui fonctionnent en mode infrastructure (cf. figure 9.3). Le `AP` est l'équivalent d'un concentrateur en réseau filaire. Lorsqu'on utilise des APs, les communications entre éléments mobiles passent toutes par ces points d'accès.

Par exemple, en mode infrastructure, les éléments B et C ne peuvent pas voir directement D, E et F, mais ces éléments peuvent tout de même communiquer entre eux en utilisant les deux point d'accès (Point d'accès 1 et point d'accès 2).

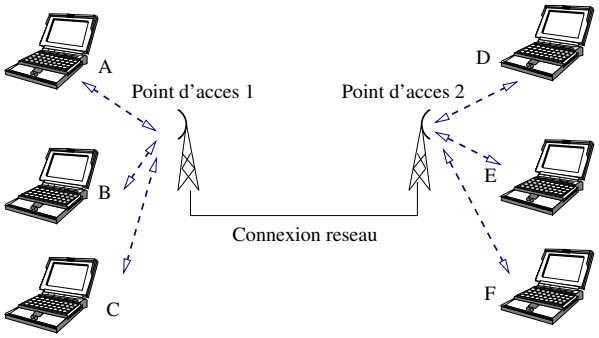

FIG. 9.3 – Accès Wifi en mode infrastructure

De même, les éléments A, B et C ne peuvent pas se voir et donc utilisent le point d'accès 1 pour communiquer entre eux.

9.3.2 Les réseaux sans fil en mode ad hoc

Le réseau est composé uniquement de stations (appelées nœuds), sans aucun équipement central (le protocole est donc un protocole point-à-point). Les nœuds assurent souvent les fonctions de routage (par exemple OLSR [62]) pour acheminer l'information d'une station vers une autre (cf. figure 9.4).

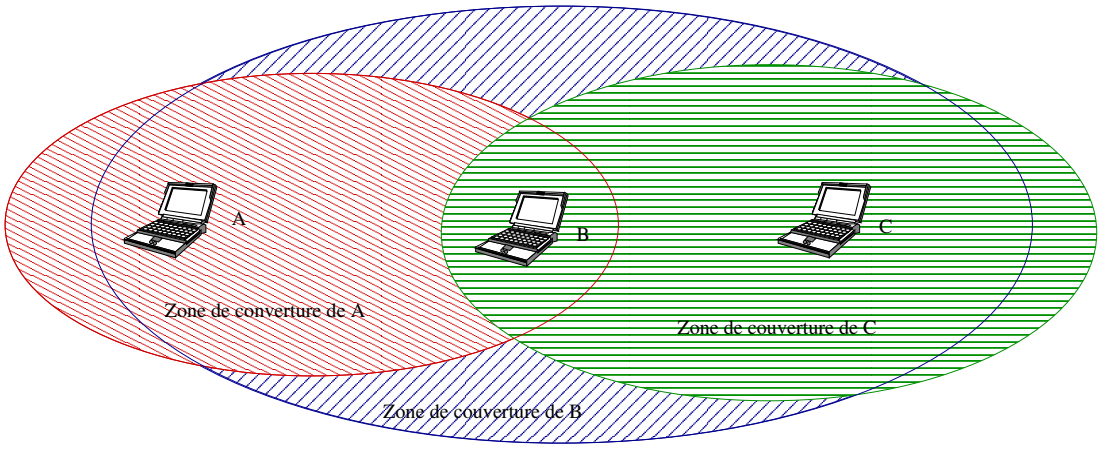

FIG. 9.4 – Accès Wifi en mode `ad hoc`

Les éléments A, B, C peuvent communiquer directement les uns avec les autres sans passer par un point d'accès. Toutefois, si A désire communiquer avec l'élément C qui n'est pas dans sa `Zone` de couverture, la communication passera par l'élément B. L'élément B joue le rôle de routeur entre les deux zones de couvertures.

Bibliographie

[1] N. Abramson. The Aloha system. In AFIPS Press, editor, *AFIPS Conference Proceedings*, volume 37, pages 281–285, Las Vegas, Nevada, 1970. 2.3.1, 2.3.2

[2] AdventNet. AdventNet SNMP tools, 1998. `http://www.adventnet.net`. 3.2.4.5, 3.2.6, 6.3.1

[3] Anand R. Tripathi and Neeran M. Karnik and Ram D. Singh and Tanvir Ahmed and John Eberhard and Arvind Prakash. Development of Mobile Agent Applications with Ajanta. Technical report, Department of Computer Science, University of Minnesota, 1999. `http://www.cs.umn.edu/Ajanta`. 3.2.4.3, 3.3.3.4

[4] Yariv Ardor and Danny B. Lange. Agent Design Patterns : Elements of Agent Application Design. In *Proc. 2nd Int'l Conf. on Autonomous Agents*, pages 108–115. ACM Press, 1998. 3.3.1

[5] Asynchronus Transfer Mode, 1993. RFC 1483. 2.3.2

[6] I. Attali, D. Caromel, and A. Contes. Hierarchical and Declarative Security for Grid Applications. *High Performance Computing - HiPC 2003, 10th International Conference, Hyderabad, India, December 17-20, 2003, Proceedings*, 2913() :363–372, December 2003. Lecture Notes in Computer Science. 6.4.2.1

[7] F. Baude, D. Caromel, F. Huet, L. Mestre, and J. Vayssière. Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications. *11th IEEE High Performance Distributed Computing, HPDC-11 (HPDC'20)*, 2002. 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002), 23-26 July 2002, Edinburgh, Scotland, UK. 5.2.3, 5.3.4

[8] Françoise Baude, Denis Caromel, Fabrice Huet, and Julien Vayssière. Objets Actifs Mobiles et Communicants. *Technique et Science Informatiques*, 21(6) :823–849, 2002. 3.2.4.3, 4.3.1.2

[9] P. Bellavista, A. Corradi, and C. Stefanelli. An Open Secure Mobile Agent Framework for Systems Management. *Journal of Network and Systems Management*, 7(3), september 1999. `http://citeseer.nj.nec.com/bellavista99open.html`. 3.2.3, 3.2.4.5, 3.3.2.2

[10] P. Bellavista, A. Corradi, and C. Stefanelli. How to Monitor and Control Resource Usage in Mobile Agent Systems. In *3 rd IEEE Int. Symp. on Distributed Objects and Applications (DOA'01), Italy*. IEEE Computer Society Press, September 2001. `http://citeseer.nj.nec.com/bellavista01how.html`. 3.2.2, 3.2.6

[11] A. Bieszczad, B. Pagurek, and T. White. Mobile Agents for Network Management. *IEEE Communications Surveys 1, 1*, 1998. 3.1

[12] Raouf Boutaba and Jin Xiao. Network Management : State of the Art. In Lyman Chapin, editor, *Communication Systems : The State of the Art (IFIP World Computer Congress)*, pages 127–146. Kluwer, 2002. `www.ifip.tu-graz.ac.at/TC6/events/WCC/WCC2002/papers/Boutaba.pdf`. 2.5.2.2

[13] Yuri Breitbart, Minos N. Garofalakis, Cliff Martin, Rajeev Rastogi, S. Seshadri, and Abraham Silberschatz. Topology Discovery in Heterogeneous IP Networks. In *IEEEE/INFOCOM 2000, Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 265–274, 2000. `http://citeseer.nj.nec.com/breitbart00topology.html"`. 3.4.1, 3.4.1.2, 3.4.3.2

[14] D. Milojicicand M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF : The OMG Mobile Agent System Interoperability Facility. In *Second International Workshop on Mobile Agents 98 (MA'98)*, September 1998. 3.2.2, 3.2.5

[15] Jean-Pierre Briot and Rachid Guerraoui. Objets pour la programmation parallèle et répartie : intérêts, évolutions et tendances. *Technique et Science Informatiques (TSI)*, 15(6) :765–800, June 1996. 4.2.2

[16] Jamie Cameron. A web-based system administration tool for Unix. USENIX Annual Conference, June 2000. `http://www.webmin.com/`. 2.2.3

[17] D. Caromel, W. Klauser, and J. Vayssiere. Towards Seamless Computing and Metacomputing in Java. *pp. 1043–1061 in Concurrency Practice and Experience, 10(11–13)*, 1998. 4.2.2

[18] Denis Caromel. Toward a method of object-oriented concurrent programming. *Communications of the ACM*, 36(9) :90–102, sep 1993. 4.2.2

[19] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. RFC 1441. Introduction to SNMP V2, April 1993. `http://www.faqs.org/rfcs/rfc1441.html`. 2.4.1.8

[20] J. Case, D. Partain, and B. Stewart. RFC 3410. SNMP V3, December 2002. `http://www.faqs.org/rfcs/rfc3410.html`. 2.4.1.8

[21] Marco Castaldi, Antonio Carzaniga, Paola Inverardi, and Alexander L. Wolf. A Lightweight Infrastructure for Reconfiguring Applications. volume

2649/2003, pages 231–244, Februrary 2004. http://www.cs.colorado.edu/~carzanig/papers/cciw_scm11.pdf. 8.2

[22] Omar CHERKAOUI. *Standards pour la gestion des réseaux et des services*. Olivier Festor and André Schaff, 2003. 5.4.2

[23] Graphical network viewer : Cisco Works 2000. http://www.cisco.com/warp/public/cc/pd/wr2k/. 2.5.2.1

[24] CMOT. The Common Management Information Services and Protocol over TCP/IP. RFC 1095. http://rfc.sunsite.dk/rfc/rfc1095.html. 4

[25] DHCP. Dynamic host configuration protocol. rfc 1531. http://www.ietf.org/rfc/rfc1531.txt. 5.2.1.2

[26] DNS : Serveur de nom de domaine. RFC 1034. 5, 3.4.2.2

[27] Etherape. Graphical network viewer. http://etherape.sourceforge.net/. 2.5.1

[28] S. Fontanini, J. Wainer, V. Bernal, and S. Marangon. Model Based Diagnosis in LANs. In *IEEE Workshop on IP Operations and Management*, pages 121–5, Dallas, October 2002. http://www.ic.unicamp.br/~wainer/papers/ipom2002.pdf. 3.4.1.2

[29] TeleManagement Forum. ITU-T [X.700] OSI Management framework. http://www.nmf.org. 2.2

[30] Germán Goldszmidt, Yechiam Yemini, and Shaula Yemini. Network Management : The MAD Approach. In *Proceedings of the IBM/CAS Conference*, Toronto, Canada, October" 1991. 3.2, 3.2.1

[31] R. S. Gray. Agent Tcl : A flexible and secure mobile-agent system. In M. Diekhans and M. Roseman, editors, *Fourth Annual Tcl/Tk Workshop (TCL 96)*, pages 9–23, Monterey, CA, 1996. http://citeseer.ist.psu.edu/gray97agent.html. 3.2.2

[32] Frédéric Guidec, Yves Mahéo, and Luc Courtrai. "aa java middleware platform for resource-aware distributed applications". In *Second International Symposium on Parallel and Distributed Computing*, Ljubljana, Slovenia, October 2003. http://citeseer.ist.psu.edu/gray97agent.html. 5.3.4, 8.2

[33] Delbert Hart, Mihail Tudoreanu, and Eileen Kraemer. Mobile agents for monitoring distributed systems. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 232–233, Montreal, Canada, 2001. ACM Press. http://citeseer.nj.nec.com/hart01mobile.html. 3.4.1.2

[34] P.W Chen H.C Lin, S.C Lai and H.L Lai. Automatic Topology Discovery of IP Networks, january 2000. IEICE Transactions on Information and Systems. 3.4.2.2, 3.4.3.2

[35] Hewlett-Packard. HP Network Node Manager. http://www.openview.hp.com/solutions/nim/index.html. 1.1, 2.2.1, 2.5.2.1, 3.4.2.1, 5.3.2.1

[36] Host Resource MIB, march 2000. RFC 2790. 6.3.1.2

[37] IBM. Nways, 2000. http://www.networking.ibm.com/. 1.1

[38] Internet Control Message Protocol, 1981. RFC 792. 2.3.3

[39] Internet Protocol, 1981. RFC 791. 2.3.3

[40] Julio Guijarro and Manuel Monjo and Patrick Goldsack. Framework for managing large scale component-based distributed applicaions using JMX. http://www.j2eeolympus.com/J2EE/JMX/JMXEJB.jsp. 8.2

[41] R. Kavasseri and B. Stewart. Rfc 2981/ event mib, October 2000. http://www.faqs.org/rfcs/rfc2981.html. 2.4.3.2

[42] R. Kavasseri and B. Stewart. Rfc 2982 / distributed management expression mib, October 2000. http://www.faqs.org/rfcs/rfc2982.html. 2.4.3.2

[43] R. Kavasseri and B. Stewart. Rfc 3014/ notification log mib, November 2000. http://www.faqs.org/rfcs/rfc3014.html. 2.4.3.2

[44] LDAP. Lightweight Directory Access Protocol. RFC 1177, 1995. http://www.ldapcentral.com/. 4.4.2.1

[45] D. Levi and J. Schoenwaelder. Rfc 2591 / scheduling management operations mib, November 2000. http://www.ietf.org/rfc/rfc2591.txt. 2.4.3.1

[46] Bruce B. Lowekamp, David R. O'Hallaron, and Thomas Gross. Topology Discovery for Large Ethernet Networks. In *Proceedings of SIGCOMM 2001*, pages 237–248. ACM, August 2001. 3.4.1.2, 3.4.3.2

[47] Line Printer Daemon Protocol, august 1990. RFC 1179. 1

[48] M. Forssen and F. Cusack. SSH Connection Protocol. http://openbsd.appli.se/openssh/txt/draft-ietf-secsh-auth-kbdinteract-01.txt. 7.3.2

[49] J.P. Martin-Flatin, S. Znaty, and J.P. Hubaux. A Survey of Distributed Enterprise Network and Systems Management Paradigms. In *Journal of Network and Systems Management)*, volume 7-1, pages 9–26, 1999. http://www.cstp.umkc.edu/jnsm/vols/vol07n1.html. 2.5.2.2

[50] MIAMI. Mobile Intelligent Agents for Managing the Information Infrastructure . http://www.cordis.lu/infowin/acts/analysys/products/thematic/agents/ch3/miami.htm. 3.2.2

[51] Sun MicroSystems. Java developpement kit - version 1.2. 3.1

[52] Sun MicroSystems. Java Management Extensions. http://java.sun.com/products/JavaManagement/wp/. 9.2

[53] Sun MicroSystems. Jini Network Technology. http://wwws.sun.com/software/jini/. 4.4.3.1, 5.3.1, 9.1

[54] Nikos Migas, William J. Buchanan, and Kevin A. Mc Aartney. Hierarchical network management : a scalable and dynamic mobile agent-based approach . volume 38, pages 693–711, April 2002. 8.1

[55] Nikos Migas, William J. Buchanan, and Kevin A. Mc Aartney. Mobile Agents for Routing, Topology Discovery, and Automatic Network Reconfiguration in Ad-Hoc Networks . In *IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'03)*, April 2003. 7.2.2.4

[56] Multi Router Traffic Grapher. `http://people.ee.ethz.ch/~oetiker/webtools/mrtg/`. 2.2.1, 2.5.1

[57] Nagios. An open source host, service, and network monitoring program. `http://www.nagios.org`. 5.3.2.1

[58] Nöel de Palma. Services d'Administration d'Applications Réparties. Thèse soutenue à l'Université Joseph Fourier. `http://sardes.inrialpes.fr/people/depalma/perso.html`. 8.2

[59] Wireless Local Area Network. Ieee 802.11. `http://grouper.ieee.org/groups/802/11/`. 7.2.2.1

[60] The Institute of Electronics and Inc Electronics Engineers. IEEE Standards for Local Area Network : Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications. Technical report, The Institute of Electronics and Electronics Engineers, Inc., New York, 1985. 2.3.2

[61] The Institute of Electronics and Inc Electronics Engineers. IEEE Standards for Local Area Networks : Token Ring Access Method and Physical Layer Specifications. Technical report, The Institute of Electronics and Electronics Engineers, Inc., New York, 1985. 2.3.2

[62] OLSR. Optimized Link State Routing. `http://hipercom.inria.fr/olsr/`. 9.3.2

[63] M.J. O'Mahony, D. Gavalas, D. Greenwood, and M. Ghanbari. An Infrastructure for Distributed and Dynamic Network Management based on Mobile Agent Technology. *Proc. IEEE International Conference on Communications (ICC'99)*, 2 :1362–1366, June 1999. `http://citeseer.nj.nec.com/context/1055077/0`. 3.2.4.4

[64] B. Pagurek, Y. Wang, and T. White. Integration of Mobile agents with SNMP : Why and How. In *IEEE/IFIP Network Operations and Management Symposium, NOMS 2000, Honolulu*, April 2000. `http://www.sce.carleton.ca/netmanage/publications.html`. 1.2

[65] D. Plummer. An Ethernet Address Resolution Protocol. Technical report, Symbolics Cambridge Research Center, November 1982. RFC 826. 2.3.3

[66] A. Pras and J. Schonwalder. *Editorial*, volume 7. The Simple Times, `http://www.simple-times.org/pub/simple-times/issues/7-2.html`, Novembre 1999. 2.4.3.1

[67] ProActive. INRIA, 1999. `http://www-sop.inria.fr/oasis/ProActive`. 4.1

[68] A. Puliafito and O. Tomarchio. Security Mechanisms for the MAP Agent System. In *In 8th Euromicro Workshop on Parallel and Distributed Processing (PDP2000)*, pages 84–91, January 2000. `http://sun195.iit.unict.it/~otomarch/publications.html`. 3.2.2

[69] A. Puliafito, O. Tomarchio, and L. Vita. MAP : Design and Implementation of a Mobile Agent Platform. *Journal of System Architecture*, 46(2) :145–162, 2000. 3.2.3, 3.2.4.1, 3.2.4.1, 3.2.4.3

[70] S. Sharma R. Siamwalla and S. Kashav. Discovering Internet topology. Technical report, Cornell University, Ithaca, NY 14583, July 1998. `http://www.cs.cornell.edu/skeshav/papers/discovery.pdf`. 3.4.2.2, 3.4.2.2

[71] M. Ranganathan, Laurent Andrey, Virgine Galtier, and Virgine Schaal. AGNI : Encore des Agents Mobiles ! In *Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'99)*, April 1999. `http://snad.ncsl.nist.gov/agni/agni/docs/white-paper/cfip99.ps`. 3.2.2, 3.4.2.2

[72] E. Reuter and F. Baude. A mobile-agent and SNMP based management platform built with the Java ProActive library. *IEEE Workshop on IP Operations and Management (IPOM 2002)*, pages 140–145, October 2002. Dallas. 5.3.2.2

[73] E. Reuter and F. Baude. System and Network Management Itineraries for Mobile Agents. *4th International Workshop on Mobile Agents for Telecommunications Applications, MATA*, LNCS(2521) :227–238, October 2002. Barcelona. 4.4.4

[74] Luigi Rizzo. Dummynet : a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1) :31–41, 1997. `http://info.iet.unipi.it/~luigi/ip_dummynet`. 7.2.1.1

[75] Daniel Rossier-Ramuz. Ecomobile : a Mobile Agent Ecosystem for Active Network Management. Technical report, Departement of Informatics, University of Fribourg, June 2003. `http://diuf.unifr.ch/pai/publications/internal.php`. 1.2

[76] A. Sahai and C. Morin. Enabling a Mobile Network Manager (MNM) Through Mobile Agents. In *International Workshop on Mobile Agents*. LNCS, No. 1477, Springer-Verlag, september 1998. 3.2.2

[77] Ichiro Satoh. Network processing of Mobiles Agents, by Mobile Agents, for Mobile Agents. In *3th International Workshop on Mobile Agents*

for Telecommunications Applications, MATA'01, volume 2146, pages 81–92. Lecture Notes in Computer Science (LNCS), August 2001. `http://citeseer.nj.nec.com/575168.html`. 3.2.2

[78] Ichiro Satoh. A Framework for Building Reusable Mobile Agents for Network Management. In *Proceedings of Network Operations and Managements Symposium (NOMS'2002)*, pages 51–64. IEEE Communication Society, April 2002. `http://citeseer.nj.nec.com/575168.html`. 3.2.4.1, 3.2.4.1, 3.2.4.3, 3.3.2.3, 5.3.2.1

[79] J. Schonwalder and H. Langendorfer. How to Keep Track of Your Network Configuration. In *Proceedings of the 1993 LISA VII Conference*, 1993. `http://citeseer.nj.nec.com/article/to93how.html`. 3.4.1, 3.4.2.2

[80] Disman Script Mib. RFC 2592. 2.4.3.1

[81] P. Simões, F. Boavida, and L. Silva. A Generic Management Model for Mobile Agent Infrastructures. In *Proceedings of SCI 2000 - The 4th World Multiconference on Systemics, Cybernetics and Informatics*, July 2000. 3.2.3

[82] P. Simões, R. Reis, L. Silva, and F. Boavida. Enabling Mobile Agent Technology for Legacy Network Management Frameworks. In *Proceedings of Softcom '99 - Conference on Software in Telecommunications and Computer Networks*. IEEE Communications Society, October 1999. 3.2.3

[83] P. Simões, Luis Moura Silva, and Fernando Boavida. Integrating SNMP into a Mobile Agent Infrastructure. In *Proceedings of DSOM'99 - Tenth IFIP/IEEE International Workshop on Distributed Systems : Operations and Management*, October 1999. `http://citeseer.nj.nec.com/358231.html`. 8.2

[84] Paulo Simões, Paulo Marques, Luís Silva, João Gabriel, and Fernando Boavida. Towards Manageable Mobile Agent Infrastructures. In *Proceedings of ICN'01 (International Conference on Networking)*. Springer-Verlag LNCS 2094, Colmar, France, July 2001. 3.2.2

[85] Paulo Simões, João Rodrigues, Luís M. Silva, and Fernando Boavida. Distributed Retrieval of Management Information : Is it About Mobility, Locality or Distribution. In *NOMS'2002 - Network Operations and Management Symposium)*. IEEE Communications Society, April 2002. 7.1, 7.2.1.3, 8.1

[86] Simple Network Management Protocol, 1990. RFC 1157. 2.4.1

[87] Simple Network Management Protocol standard MIB-II, 1990. RFC 1213. 1.1, 2.4.1.1, 2.4.1.4

[88] SNMP TRAPS, march 1991. RFC 1215. 6.3.1.1

[89] Markus Strasser and Kurt Rothermel. Reliability Concepts For Mobile Agents. *International Journal of Cooperative Information Systems (IJCIS)*, 7(4), 1998. `http://citeseer.nj.nec.com/270979.html`. 3.3.1

[90] Sun MicroSystems. Java Developpement Kit V 1.4. `http://java.sun.com`. 4.3.1

[91] Andrew Tanenbaum. *Réseaux : Architecture, protocole, applications, 3ème édition, Dunod*. Prentice Hall, 1999. 2.3

[92] Tcl. Tool Command Language. `http://www.tcl.tk/scripting/primer.html`. 3.2.2

[93] Transmission Control Protocol, 1981. RFC 793. 2.3.3

[94] The Grasshopper Agent Platform. `http://www.grasshopper.de`. 3.2.2

[95] Tivoli. A network management tool for enterprises and service providers. `http://www-3.ibm.com/software/tivoli/`. 2.5.2.1, 3.4.2.1

[96] User Datagram Protocol, 1980. RFC 768. 2.3.3

[97] Bill Venners. Under the hood : The architecture of aglets. *JavaWorld : IDG's magazine for the Java community*, 2(4), apr 1997. 4.2

[98] Voyager. ObjectSpace, Inc., 1999. `www.objectspace.com`. 4.2

[99] Chris Wellens and Karl Auerbach. *The Quarterly Newsletter of SNMP Technology, Comment, and Events (sm)*, volume 4 (3). The Simple Times, `http://www.simple-times.org/pub/simple-times/issues/4-3.html`, July 1996. 2.5.3

[100] K. White. Rfc 2925. definitions of managed objects for remote ping, traceroute, and lookup operations, September 2000. `http://www.faqs.org/rfcs/rfc2925.html`. 2.4.3.2

[101] Buchanan WJ, Naylor M, and Scott AV. Enhancing Network Management using Mobile Agents. In *Seventh IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2000))*, pages 218–226. IEEE Communications Society, 2000. 7.1

[102] Mark Wolfgang. Nmap hackers, november 2002. White Paper Exploring Host Discovery. 2.3.3

[103] Rich Wolski, Neil Spring, and Jim Hayes. The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing. volume 15, pages 757–768, October 1999. `http://nws.cs.ucsb.edu/publications.html`. 8.2

[104] W.Stallings. *SNMP, SNMPv2, and CMIP*. Don Mills : Addison-Wesley, 1993. 1.1, 2.4.1.1, 2.4.2, 4.4.1.2

Résumé

Depuis le début des années 1990, l'utilisation des agents mobiles dans le cadre de l'administration système et réseau est étudiée. Des plates-formes à agents mobiles répondant à cet usage ont été développées et se sont penchées en général sur les problèmes de l'interrogation des équipements actifs du réseau, de la supervision des nœuds des plates-formes, de la sécurité des nœuds visités, etc. Cependant, une contrainte opérationnelle provenant du fait que la présence d'éléments dans un réseau tend à évoluer dynamiquement n'avait pas été prise en compte. Une telle évolution requiert que les itinéraires de migration associés aux agents mobiles d'administration ne soient pas définis comme cela est fait en général, de manière statique par l'administrateur, mais qu'au contraire de tels itinéraires puissent être construits à la volée de sorte à refléter la topologie courante du réseau à administrer. Nous avons aussi constaté que les facilités de programmation obtenues par l'association d'une fonction d'administration à chacun des éléments d'un itinéraire, et ce, en fonction du type de l'élément, pourraient être bénéfique, en ce sens que cela pourrait grandement simplifier la définition d'agents mettant en œuvre de nouvelles fonctions d'administration système et réseau.

Ce travail de thèse consiste à proposer la définition d'un mécanisme de fabrication puis d'utilisation d'itinéraires dynamiques pour l'administration système et réseau. Nous validons cette définition en fournissant une plate-forme complète ainsi qu'un cadre de programmation d'agents mobiles d'administration système et réseau. Pour ce faire, nous utilisons et étendons la bibliothèque ProActive pour le calcul parallèle, réparti et mobile. En effet, ProActive offre un cadre qui simplifie grandement la programmation et le déploiement d'agents mobiles en environnement hétérogène et réparti.

Le mécanisme d'itinéraires proposé suppose la connaissance de la topologie du réseau et des éléments qui y sont connectés. Nous implantons donc un mécanisme de découverte automatique de la topologie d'un réseau au niveau 2 OSI. Nous définissons ensuite un cadre générique de programmation d'agents mobiles bâtie autour de notre mécanisme d'itinéraires pour effectuer des tâches d'administration système et réseau. Pour valider notre approche et son intégration dans le monde de l'administration système et réseau, nous fournissons des exemples concrets de l'utilisation des agents mobiles d'administration et ce pour une large gamme de configurations réseau.

`Mots-clés` : agents mobiles, administration système et réseau, itinéraire dynamique, migration, SNMP, ProActive.

Abstract

Since the beginning of 1990s, the usage of mobile agents within the framework of system and network management is considered. Platforms with mobile agents answering this use were developed and in general focus on the problem of collecting data on network elements, on supervising platform nodes, on the safety of the visited nodes, etc. However, an operational constraint relates to the fact that elements in a network tend to evolve/move dynamically ; such a constraint has not been seriously taken into account in existing platforms. Such a dynamic evolution requires that migration itineraries be associated with mobile agents for network and system management ; but such itineraries are generally defined in a static way by an administrator, alas, on the contrary, such itineraries should be built on the fly so as to reflect the current network topology to be managed. We underline that programming facilities could be obtained by associating a management function with each element in an itinerary, and this, according to the type of each element. This could be beneficial, in the sense that it could largely ease the programming of mobile agents tasks by enabling to implement new system and network management functions.

This thesis work consists in proposing the definition of a mechanism to create and then to use dynamic itineraries for system and network management. We validate this definition by providing a platform and a mobile agents programming framework for system and network management. In order to achieve this, we use and extend the ProActive library for parallel, distributed and mobile computing. Indeed, ProActive offers a framework which largely simplifies the programming and the deployment of mobile agents in an heterogeneous and distributed environment.

The proposed itineraries mechanism assumes the knowledge of the network topology and how its elements are interconnected. Thus, we design and implement an automatic discovery mechanism of the OSI level-2 topology of the network to manage. Then, we define a generic framework for mobile agents programming based upon our itineraries mechanism so as to carry out system and network management tasks. To validate our approach and its integration in the system and network management world, we provide several usage scenario of mobile agents, and this for a broad range of network configurations.

`Keywords` : mobile agents, system and network management, dynamic itinerary, migration, SNMP, ProActive.